

**HANNES BAJOHR,
MARKUS KRAJEWSKI (HG.)**

QUELLCODEKRITIK

Zur Philologie von Algorithmen

August Verlag

INHALT

Quellcodekritik. ReadMe1st.txt 9
Hannes Bajohr und Markus Krajewski

I. Das sequenzielle Paradigma

Critical Code Studies. Ein Manifest 27
Mark C. Marino

Kulturtechnik Programmieren.
Quellcode kritisieren. Drei Beispielszenarien 63
Markus Krajewski

Wie liest man 100.000 Zeilen Code? 87
Till A. Heilmann

Programmieren mit Copilot.
Über Grenzen der Automatisierung und
Formen von Subjektivierung 127
Dan Verständig

II. Das konnektionistische Paradigma

Spec Acts. Formales Lesen in rekurrenten
neuronalen Netzen 155
Matthew Kirschenbaum

Dumme Bedeutung. Künstliche Intelligenz und artifizielle Semantik <i>Hannes Bajohr</i>	197
Verkettete Textualität <i>Tyler Shoemaker</i>	217
Mit den Toten chatten. Die Hermeneutik von Thanabots <i>Leah Henrickson</i>	245
Über die Gefahren stochastischer Papageien. Können Sprachmodelle zu groß sein? 🐦 <i>Emily Bender, Timnit Gebru, Angelina McMillan-Major und Shmargaret Shmitchell</i>	277

QUELLCODEKRITIK. README1ST.TXT

Hannes Bajohr und Markus Krajewski

Algorithmen, so könnte man in Anlehnung an einen Satz Friedrich Kittlers sagen, bestimmen unsere Lage. Von *bubble sort* und Googles PageRank über Kreditwürdigkeitsprüfungen bis hin zum *predictive policing* – die Logik der Algorithmen greift auf Schritt und Tritt in unser Leben ein. Einige arbeiten undurchsichtig und sind gut gehütete Geschäftsgeheimnisse. Bei anderen bemüht man sich um Transparenz und folgt einer Ethik des *open source* und der Rechenschaftspflicht. In allen Fällen ist jedoch ein nicht unerheblicher Aufwand erforderlich, um die Quellcodes zu verstehen, in denen Algorithmen normalerweise niedergelegt sind. Das ist auch eine Herausforderung für die Geisteswissenschaften, von denen nur ein kleiner Teil über die nötige, wie Annette Vee es nennt, *coding literacy* verfügt, um Algorithmen mit Verständnis zu lesen und zu interpretieren, zu kommentieren und weiterzuschreiben.¹

Die Gründe, aus denen Code kein unproblematischer Gegenstand klassischer Lektüre ist, sind zahlreich. In den allermeisten Fällen ist er seinen potentiellen Leser:innen schlicht unzugänglich. Sie stehen, wenn sie ein Programm öffnen, nicht mehr dem Quellcode gegenüber, sondern einer ausführbaren Datei, die im Binärformat vorliegt. Dieser Maschinencode ist für Menschen praktisch nicht zu durchschauen. Zwar kann ein solches kompiliertes Programm über ein sogenanntes *reverse engineering* in eine Assemblersprache (bei JAVA sogar in den vollständigen ursprünglichen Quelltext ohne Kommentare) zurückübersetzt werden. Aber eine solche Transformation ist höchstens Code *a posteriori*, ein nachträgliches Produkt. Dessen

¹ Annette Vee, *Coding Literacy. How Computer Programming is Changing Writing*, Cambridge, MA: MIT Press 2017.

bloße Auflistung der allerbasalsten Kommandos und Datenstrukturen ist daher zumeist weit entfernt vom ursprünglichen Quellcode, der in den meisten Fällen in einer höheren Programmiersprache wie C, C++ oder Python geschrieben ist.

Immer öfter aber ist das, was man den Code *a priori* nennen könnte, doch frei zugänglich. Sei es, weil Entwickler:innen ältere Programme samt Quellcode in die Öffentlichkeit entlassen, sei es, weil sie bereits von vornherein unter verschiedenen Open-Source-Lizenzen über reichhaltige Online-Repositoryen wie etwa Microsofts github.com weitergegeben werden. Aber auch dieser frei verfügbare Code *a priori* ist keinesfalls trivial zu lesen. Denn auch Quellcode, der nicht hinter Barrieren versteckt ist oder bewusst unlesbar gemacht wurde, widersetzt sich Lesenden, die keine technische Expertise zu seiner Lektüre mitbringen – sei diese in einem Informatikstudium erworben oder auch nur bloßes Amateurwissen. Dabei ist es heute dringender denn je, auch außerhalb dieser Gruppen zeitgemäße Medienkompetenz in Form zuverlässiger *coding literacy* zu fördern.

Auch in den Geistes-, Sozial- und Rechtswissenschaften sollten Wissenschaftler:innen die Fähigkeiten erwerben und weitergeben, die notwendig sind, um nicht nur komplexe philosophische, juristische und literarische Sachverhalte in Textform zu erforschen, sondern eben auch Codes zu verstehen und sie aus einer kritischen Perspektive einordnen zu können. Denn obwohl Code eine Art von Sprache ist und von Menschen und Maschinen gleichermaßen geschrieben und gelesen wird, vermuten die philologischen Fakultäten Algorithmen zumeist außerhalb ihres Fachbereichs und ihrer Kompetenzen. Selbst die Medienwissenschaft, unsere eigene Disziplin, die über die Instrumente und Konzepte verfügen sollte, um sich mit ihnen zu befassen, bleibt oft zurückhaltend und zieht es vor, nur abstrakt über algorithmische Macht zu sprechen.

Die Behauptung, Algorithmen seien abstrakte Artefakte, ist an sich nicht falsch. Aus kulturhistorischer Sicht bezeichnen

Algorithmen eine Rechenregel, ein Kalkül, das aus einer linearen Abfolge von Befehlen besteht. Von einem klar definierten Startpunkt aus wird er Schritt für Schritt ausgeführt, um eine bestimmte Aufgabe oder ein Problem zu lösen. Ein Algorithmus kann daher als eine in sich geschlossene Kette von Kommandos oder als eine Operationskette betrachtet werden. Ist diese Kette in einer Programmiersprache geschrieben, spricht man von Code (wobei dieser neben Algorithmen etwa auch Datenstrukturen, Bibliotheken und andere Ressourcen umfassen kann).

Um wirksam zu sein, benötigt Code einen Computer, der ihn ausführt. Wohl keine Technik kann nur für sich allein funktionieren, ohne die Mitwirkung anderer. Gerade deshalb ist es naheliegend zu sagen, dass Algorithmen nicht allein abstrakte Gebilde sind, sondern auch eine soziale Wirklichkeit besitzen, weil ihre Funktionen immer an den Schnittstellen zwischen Maschinen und Menschen angesiedelt sind: Sie verarbeiten nicht nur datentechnisch menschliche Eingaben, sondern in den allermeisten Fällen entwerfen Softwareentwickler:innen – im Moment noch überwiegend Menschen – die konkreten algorithmischen Abläufe. Gleichzeitig modellieren die praktisch umgesetzten Algorithmen ihre Nutzer:innen, beispielsweise wenn diese beim Gebrauch einer bekannten Suchmaschine die *cookies* akzeptieren und infolgedessen mithilfe vieler Messpunkte individuelle Eigenschaften wie Computerkonfiguration, Browser, Betriebssystem, Sprache etc. erfasst werden.

Darüber hinaus könnten Algorithmen aber auch in einem anthropologischen Sinne als Operationsketten, als *chaîne opératoire*,² beschrieben werden. In diesen Operationsketten werden verschiedene Akteur:innen – menschlicher wie technischer Art – zu einer temporären Handlungseinheit zusam-

² André Leroi-Gourhan, *Hand und Wort. Die Evolution von Technik, Sprache und Kunst*, Frankfurt a.M.: Suhrkamp 1965/1980, S. 150f. und S. 275–280.

mengefasst, die erst durch das Zusammenspiel von Objekten, Medien und Personen eine bestimmte Wirkung erzielt. Eine Operationskette organisiert das Zusammenspiel einer manuellen Geste (Technik), eines Werkzeugs (*l'outil*) und symbolischer Operationen (Sprache), um eine Handlung zu vollziehen. Operationsketten sind daher weder nur Technik und Prozess, sondern eben auch soziale Konstrukte mit sozialen Auswirkungen, weil sie die Interaktion zwischen menschlichen und nicht-menschlichen Akteuren konstituieren und begleiten.

Diese Charakterisierung versteht Algorithmen als soziale Akteure. Als Handlungskette kann die Ausführung eines Algorithmus nicht nur virtuelle und soziale, sondern auch physisch-materielle Wirkungen erzeugen. In einer solchen anthropologischen Perspektive verwandeln Algorithmen als Operationsketten bestimmte Ideen in konkrete Handlungen, indem sie spezifische Werkzeuge (z.B. Filter) oder Techniken (in der Hardware ebenso wie in der Software) verwenden.

Code handelt demnach als Akteur in komplexen Operationsketten, und seine Auswirkungen sind nicht auf den Computer allein beschränkt, sondern spielen in gesellschaftliche, rechtliche und wirtschaftliche Beziehungen in der realen Welt hinein. Derart situierte Algorithmen sind, anders gesagt, in soziale Beziehungen eingebettet und agieren als deren Konstrukte an den Schnittstellen zwischen Mensch und Maschine, werden in konkreten Plattformen implementiert und von realen Menschen geschrieben. Als Grenzobjekte, deren Effekte sich auf eine Vielzahl von sozialen Feldern auswirken, kann ihr Tun verfolgt und ihre Interaktion mit der Realität bewertet werden – ohne jemals *in* diese Objekte hineinzuschauen.

Aber auch das ist noch eine Art Vogelperspektive, die nur ein Teil einer wirklich umfassenden Codephilologie sein kann. Eine solche Makrokomplexität kann nur dann sinnvoll analysiert werden, wenn die Funktionsweise ihrer Objekte auch auf der *Mikroebene* verstanden wird. Mit anderen Worten: Es ist

erforderlich, Code auf den Grund zu gehen und ihn tatsächlich *zu lesen*. Das heißt, die Texte von höheren Programmiersprachen einer Lektüre zu unterziehen.

Dass eine solche Lektüre gewisse Probleme mit sich bringt, die über den Rahmen der klassischen Hermeneutik hinausgehen, ist offensichtlich, und David Berry, Mark C. Marino oder Kevin Brock haben dies hinreichend deutlich gemacht.³ Codes sind schließlich ganz besondere Sprachartefakte. Sie führen Aktionen aus, wenn sie ausgeführt werden, was Florian Cramer dazu veranlasste, sie in einer Linie mit Zaubersprüchen und anderen „selbstaufführenden Texten“ zu sehen.⁴ Mit einer metonymischen Charakteristik Kittlers gesprochen erhält die Schrift damit ein ganz eigenes, neues Handlungsvermögen: „Die ENTER-Taste hat eine Macht erlangt, die den Wortsinn von Poesie, nämlich ‚machen‘, im Unterschied zu aller Poesie oder Literatur der Geschichte erstmals einlöst.“⁵

Potenziert der Code die pragmatische Seite der Sprache (ohne im wirklichen Sinne Sprechakt zu sein),⁶ reduziert er seine expressive Dimension auf die reine Denotation. Er ist, wie Charles Taylor es nannte, „depiction without expression“⁷ – ein Status, den keine sogenannte „natürliche Sprache“ erreichen kann, auch wenn Code für gewöhnlich einen nicht geringen Anteil an natürlicher Sprache beinhalten muss, um lesbar zu bleiben. Weil er zunächst keinerlei konnotative oder referenzielle Dimension hat, treibt er die Konventionalität von

³ David M. Berry, *The Philosophy of Software. Code and Mediation in the Digital Age*, Basingstoke: Palgrave Macmillan 2011; Kevin M. Brock, *Rhetorical Code Studies: Discovering Arguments in and around Code*, Ann Arbor: University of Michigan Press 2019; Mark C. Marino, *Critical Code Studies*, Cambridge, MA: MIT Press 2020.

⁴ Florian Cramer, *Exe.cut[up]able statements. Poetische Kalküle und Phantasmen des selbstaufführenden Texts*, München: Fink 2011.

⁵ Friedrich Kittler, *Daten → Zahlen → Codes. Vortrag an der Hochschule für Grafik und Buchkunst*, Leipzig: Institut für Buchkunst 1998, S. 20.

⁶ Vgl. Martin Bartelmus und Alexander Nebrig (Hg.), *Schriftlichkeit. Aktivität, Agentialität und Aktanten der Schrift*, Bielefeld: Transcript 2022.

⁷ Charles Taylor, „Theories of Meaning“, in: ders., *Human Agency and Language. Philosophical Papers I*, Cambridge: Cambridge University Press 1985, S. 247–292, hier S. 267.

Sprache ins Extrem: Alle Ähnlichkeiten zu natürlichsprachigen Wörtern sind in Programmiersprachen reine Bequemlichkeit zum Erlernen für menschliche User:innen. Damit sind sie zwar auf Lesbarkeit ausgelegt, die ist aber von der Regel- und Anweisungshaftigkeit ihrer Ausführung abgekoppelt – Lektüre- und Ausführungsfunktion klaffen auseinander. Der Programmierpapst Donald Knuth soll daher gesagt haben: „Programmieren ist die Kunst, einem anderen Menschen zu sagen, was der Computer tun soll.“⁸

Und schließlich wird Code auf ansonsten träge Datenstrukturen angewandt. Damit steht eine Philologie von Quellcodes vor Text, der auf einen anderen Text einwirkt und damit eine doppelte, wenn nicht dreifache Komplexität aufwirft: Nicht nur stehen menschliche in Konkurrenz zu nichtmenschlichen Lesenden, wie Compilern oder Datenübertragungsprotokollen; der Text steht auch in Konkurrenz zu und in Interaktion mit sekundärem Text, auf den er zugreifen muss, um funktionsfähig zu sein; und er muss funktionsfähig sein, um für Leser:innen zugänglich zu sein. Kurz: In Code niedergelegte Algorithmen sind also gleichzeitig *mehr* und *weniger* als natürliche Sprache. Und sie erfordern eine Philologie, die in der Informatik ebenso zu Hause ist wie in der Textkritik.

Der vorliegende Band geht von dieser Annahme aus. Er versammelt Beiträge einer Reihe von Wissenschaftler:innen aus verschiedenen Disziplinen mit je eigenen Perspektiven auf Algorithmen und ihre Lesbarkeit, um drei Fragen zu diskutieren: 1) Wie kann Code verstanden werden? 2) Wie kann er verwaltet und gepflegt werden? 3) Und wie kann man dazu beitragen, die in weiten Teilen der Geisteswissenschaften immer noch fehlende Codekompetenz zu fördern? Oder anders

⁸ Ein unbelegter Aphorismus, der immer wieder zitiert wird, etwa in: Selma Takir, „Reading CS Classics“, in: *Communications of the ACM* 55:4 (2012), S. 32–34, hier S. 33. Mark C. Marino zitiert in seinem Aufsatz einen sinnähnlichen, aber weniger eingängigen Ausspruch, siehe S. 35.

gesagt: Es geht um die Hermeneutik, die Philologie und die Didaktik von Algorithmen.

Eine *Hermeneutik* von Algorithmen muss, wie betont, die soziale Realität von Quellcodes ebenso berücksichtigen wie ihre textuelle Mikrologie; sie vermischt die interne Bedeutung eines Codes – den Verweis auf seine eigenen Strukturen, Traditionen und kanonischen Regelmäßigkeiten – mit der Macht eines jeden Textes, stets über sich selbst hinauszudeuten und sich in das soziale Leben von Individuen und Gruppen, die Aktivitäten von Volkswirtschaften und die Regeln und Regelbefolgung von Staaten, juristischen Systemen und staatsähnlichen Akteur:innen einzufügen. Eine solche Hermeneutik muss eine Vorstellung vom ontologischen Status der Algorithmen sowie von ihrer sozialen, rechtlichen und wirtschaftlichen Handlungsfähigkeit haben. Sie sollte auch ein kritisches Auge auf die Frage werfen, in welchen Konstellationen ihre scheinbare Objektivität im Widerspruch zu ihrer kulturellen Einbettung steht.

Eine *Philologie* des Codes ist mehr als das. Sie macht den Code selbstreflexiv und fügt dem Text die Werkzeuge der philologischen Kritik und Pflege hinzu; sie folgt der These, dass der Quellcode so intensiv wie möglich dokumentiert werden sollte, damit die Algorithmen selbsterklärend werden. Schließlich lädt uns das performative Element des Quellcodes im Gegensatz zu einem gewöhnlichen Text auch dazu ein, in ihn einzugreifen und seinen Umfang, seine Möglichkeiten und seinen Nutzen zu erkunden und zu verändern. Textkritik und Hermeneutik werden so praktisch und interaktiv.

Hierin liegt auch im weitesten Sinne eine *Didaktik* von Code. Denn eine solche Interaktion ist eine der Hauptaufgaben der gegenwärtigen Geisteswissenschaften – eine Vertrautheit mit Programmierung als solcher zu entwickeln, die die Kluft zwischen der Informatik und den interpretierenden Disziplinen überbrückt – eine Forderung, die von programmierenden

Geisteswissenschaftler:innen seit Längerem, jedoch ohne größere Resonanz erhoben wurde.⁹ Computeralphabetismus gilt es also nach wie vor – zumindest als passive Fähigkeit – in den Geisteswissenschaften weiter zu verbreiten: Es ist eine kritische Fähigkeit, die notwendig ist, um Code zu verstehen, offenzulegen, einzuordnen, zu kontextualisieren und zu erklären, um der viel beschworenen Macht der Algorithmen zu begegnen. Um zu erkennen, wie genau Benutzer:innen von einer Software benutzt werden, ist es notwendig, das Design und die Konstruktion eines Algorithmus zu entschlüsseln, zu verstehen und kritisch aufzudecken.

Einer solchen Quellcodekritik geht es nicht zuletzt darum, dass Code Geschichte hat. Aber diese Geschichte ist noch nicht an ihrem Ende. In den vergangenen Jahren nämlich hat sich das Wesen der Programmierung verändert. Der klassische Code ist heute nicht mehr die einzige Option für die Entwicklung komplexer oder auch einfacher Applikationen. Das klassische *sequenzielle* Paradigma mit seinen Regelschritten, Verzweigungen und Loops wird heute durch das *konnektionistische* Paradigma¹⁰ – stochastische maschinelle Lernmodelle – zumindest erweitert, wenn nicht infrage gestellt. Solche KI-Modelle bestehen aus so etwas *wie* Algorithmen – wenn man unter ‚Algorithmus‘ eine Funktion versteht und darunter die Abbildung von einer Menge von Eingaben auf eine Menge von Ausgaben –,¹¹ sind aber nicht mehr in derselben Weise *lesbar* wie

⁹ Vgl. Friedrich Kittler, „Den Riß zwischen Lesen und Schreiben überwinden. Im Computerzeitalter stehen die Geisteswissenschaften unter Reformdruck“, in: *Frankfurter Rundschau*, 12. Januar 1993, S. 16, sowie ders., „Computeranalphabetismus“, in: Dirk Matejovski und Friedrich Kittler (Hg.), *Literatur im Informationszeitalter*, Schriftenreihe des Wissenschaftszentrums Nordrhein-Westfalen, Bd. 2, Frankfurt a.M.: Campus 1995, S. 237–251.

¹⁰ Vgl. zu dieser Unterscheidung: Hannes Bajohr, „Algorithmische Einfühlung. Für eine Kritik ästhetischer KI“, in: ders., *Schreibenlassen. Texte zur Literatur im Digitalen*, Berlin: August 2022, S. 131–172.

¹¹ Vgl. Juan Luis Gastaldi, „The Place of Textuality in Deep Learning Approaches to Mathematics“, 2023, https://www.giannigastaldi.com/assets/pdf/pubs/Gastaldi2023b_s.pdf [Letzter Zugriff: 15.8.2023].

beispielsweise ein Python-Code. Modelle für maschinelles Lernen sind, wie man oft sagt, *Blackboxes*.¹²

Damit meint man in der Regel, dass insbesondere die Gewichtungsmodele (*weight models*) neuronaler Netze oder anderer Arten von maschinellen Lernsystemen in ihrer Funktionsweise relativ undurchsichtig sind. Ein Standardeinwand dagegen lautet, dass das Forschungsfeld der sogenannten *explainable AI* (oder *XAI*) bereits eine Vielzahl von Imaging-Methoden aufgeboten hat, die helfen, das Innere dieser Modelle für menschliches Verständnis zu öffnen. Keines dieser Verfahren ist jedoch in der Lage, das Innenleben eines stochastischen Modells in die Regelschritte eines sequenziellen Algorithmus zu übersetzen, der schließlich wieder menschenlesbar wäre. Und es ist nicht unwahrscheinlich, dass dies im Prinzip unmöglich ist. Der Punkt ist also nicht, dass die Blackbox völlig dunkel ist, sondern dass kein noch so grelles Licht uns dieselbe Klarheit verschaffen kann, die ein sequenzieller Algorithmus ermöglicht.

Dies ruft einige der grundlegendsten Annahmen der Hermeneutik und der Philologie gleichermaßen auf: dass sie, um August Boeckh zu zitieren, „die Erkenntnis des Erkannten“¹³ seien – und sich als solche mit menschlichen kulturellen Artefakten befassen, deren Verständnis eine Rückverfolgung der Praktiken und Kenntnisse ist, die in sie eingeflossen sind; das trennt sie von den Naturwissenschaften, deren Aufgabe nur eine funktionale Erklärung, aber niemals ein Verstehen sein kann. Diese Unterscheidung kann auf den Unterschied zwischen dem klassischen, sequenziellen Code und den neuen, konnektionistischen Modellen des maschinellen Lernens

¹² Vgl. Markus Krajewski, „Source Code Criticism. On Programming as a Cultural Technique and its Judicial Linkages“, in: Laurence Diver und Mireille Hildebrandt (Hg.), *The Legal Effect of Code-Driven Law. Journal of Cross-Disciplinary Research in Computational Law 2024* (in Vorbereitung).

¹³ August Boeckh, *Enzyklopädie und Methodenlehre der philologischen Wissenschaften*, hg. v. Ernst Bratuscheck, Leipzig: Teubner 1877, S. 18, Orthografie angepasst.

übertragen werden: Bei dem einen geht es um Verstehen, bei dem anderen um Erklären. Neuronale Netze als Forschungsobjekte sind durch eine Außenperspektive bestimmt – sie werden nicht anders untersucht als Gehirnstrukturen oder Sterncluster; der traditionelle Code hingegen lädt zu einer hermeneutischen, zu einer Innenperspektive ein und ist auf Verstehen ausgerichtet. *Verum ipsum factum*, der Grundsatz, der den Unterschied zwischen Naturalismus und Hermeneutik ausmacht, gilt hier: Ein von Menschen geschriebener Code wird immer anders gelesen werden als ein maschinell erstelltes Gewichtungsmodell – ungeachtet aller bildgebenden Verfahren der *explainable AI*.

Bemerkenswert bleibt dabei jedoch, dass die Herstellung der Blackboxes, die Produktion ihrer prinzipiellen Undurchschaubarkeit wiederum durch sequenziellen Code angeleitet oder programmiert wird. Keine Blackbox, etwa ein großes Sprachmodell (*large language model*, LLM), wie es ChatGPT zugrunde liegt, kann ohne das Skript in einer „klassischen“ Programmiersprache erstellt werden. Auf dieser übergeordneten Ebene fusionieren die beiden Paradigmen in einer solcher Weise, dass sich erst hier die Bedingungen des zu Erklärenden einsehen und nachvollziehen lassen. Es ist auf dieser Ebene der Skripte, der Rezepturen gleichsam, wo sich die Ausgangslage zeigt, also etwa die herangezogenen Korpora eines LLMs, der Fundus der Bilder, auf dem die Version eines Bildgenerators wie *midjourney* trainiert wurde, oder jene Teile des Internets, die in ihrer Bandbreite zwischen mäßiger Qualität und hervorragender Strukturiertheit zur Ausbildung von Weltwissen einer solchen Blackbox eingebunden wurden. Erst durch eine explizite Listung dieser Quellen im Quellcode und die Offenlegung dieser Referenzen lässt sich das Training eines Modells einigermaßen transparent und damit kritisierbar machen – auch wenn sich das Modell intern mit seiner Versammlung von Milliarden

numerischer Vektoren weiterhin einer vollen und direkten Lesbarkeit entzieht.¹⁴

Gemäß der Unterscheidung der beiden Paradigmen von sequenziellen und konnektionistischen Codes haben wir den Band strukturiert. Der erste Teil beschäftigt sich mit klassischen Befehlsfolgen, mit sequenziellen Algorithmen, der zweite diskutiert die Herausforderungen des konnektionistischen Paradigmas im maschinellen Lernen. Diese Unterscheidung, im Rahmen eines breiteren Codealphabetismus verstanden, geht dabei auch weit über das hinaus, was die *Critical Code Studies* traditionell verfolgt haben.

Den Auftakt macht Mark C. Marino mit seinem Manifest zur Methodik der *Critical Code Studies*. Dieser programmatische Aufruf zu einem philologischeren Umgang mit Code erschien – in einer früheren Version – bereits 2006 und konturiert nicht nur eine neue Methode an der Schnittstelle von Geisteswissenschaften und Computer Science. Vielmehr legt das Manifest ebenso systematisch wie einladend die Entwicklungsmöglichkeiten und Erkenntnisziele dar, wenn es darum geht, Code mit den Mitteln der literarischen Kritik, also philologisch zu lesen. Im Kern fordert dieser Ansatz dazu auf, Befehle auf ihre Verweisungszusammenhänge, auf ihre verborgenen Bedeutungen, auf das Spiel von Sprache und Metaphern, auf Ambiguitäten und Assoziationen, kurz, Codes auf ihre Rhetorizität und Poetik hin zu lesen.

Darauf aufbauend, zugleich aber auch abgrenzend und erweiternd verstehen wir die Tätigkeit der Code-Entwicklung in den Geisteswissenschaften – das Programmieren – als eine

¹⁴ Vgl. dazu sowie zur Ausweitung dieser Verfahren zu einem neuen Feld, den Critical AI Studies: Rita Raley und Jennifer Rhee, „Critical AI: A Field in Formation“, in: *American Literature* 95:2 (2023), S. 185–204.

genuine Kulturtechnik, die Verstehen vor allem durch Weiter-schreiben, durch Kommentieren und Einordnen erzielen kann. Der Beitrag von Markus Krajewski führt diesen Ansatz der Quellcodekritik aus der Perspektive der Kulturtechnikfor-schung anhand von drei Beispielen an verschiedenen Code-Fragmenten exemplarisch vor, um mit der Praktik des Kommen-tierens die epistemischen Vorzüge und Erkenntnispotentiale zu demonstrieren.

Till A. Heilmann stellt sich dem Problem der Unlesbarkeit von Code aufgrund seiner schieren Masse – selbst bei einem so alten Programm wie Photoshop 1.0.1. von 1990, das vom Her-steller Adobe als Code *a priori* öffentlich zur Einsicht freigege-ben wurde. Die mehr als 100.000 Zeilen sind für Heilmann ein exemplarischer Fall für den Versuch, große Mengen von Code für Geistes- und Kulturwissenschaftler:innen ohne tiefere Pro-grammierkenntnisse les- und analysierbar zu machen. Als Ansatz wählt er ein *distant reading* des Quelltexts, also eine oberflächliche, oft statistische Analyse mithilfe von Unix-Kom-mandozeilenwerkzeugen. Der Code wird schrittweise maschi-nell gelesen, transformiert und reduziert, um sein Panorama zu entfalten und erste Einsichten zu generieren. Dabei erweist er sich trotz seiner Funktionalität erneut als soziokulturell geprägt in seinen Sprachen, Paradigmen, Kommentaren und Benennungen.

Photoshop 1.0 konnte noch von einer einzigen Person – Tho-mas Knoll – geschrieben werden. Heute sind dagegen riesige Teams am Werk, deren Kommunikation untereinander meist über komplexe Protokolle und Dokumentationen geregelt ist. Aber nicht nur Menschen coden miteinander, sondern auch Maschinen mit Menschen. Dan Verständig untersucht in sei-nem Beitrag, wie Coding-Assistenzsysteme auf KI-Basis wie Codex und Copilot Programmieraufgaben teils automatisiert übernehmen und als *pair programmer* zusammen mit mensch-lichen Coder:innen agieren. Verständig betrachtet diese Pro-

grammiererfahrung aus didaktischer, lerntheoretischer und ethischer Perspektive und plädiert dafür, die Performativität und die subjektivierende Kraft des Programmierens in ihren Unbestimmtheiten, Leerstellen und ästhetische Erfahrungen ernst zu nehmen.

Verständig leitet mit der Verbindung von KI und Programmierung in den zweiten Teil des Bandes über, in dem zwar immer noch klassischer Code die Voraussetzung maschinellen Lernens ist, ein voll trainiertes KI-Modell aber eben nicht mehr auf dieselbe hermeneutische Weise gelesen werden kann wie ein Programmquelltext.

Matthew Kirschenbaum meint in seinem Beitrag, es bleibe hier allein die Option einer formalen Lektüre übrig, wobei er sich mit einem RNN (*recurrent neural network*) auseinandersetzt, einem Vorläufer des Transformers, der Claude, ChatGPT und Gemini (vormals: Bard) zugrunde liegt. Der Vektorraum dieses „kleinen großen“ Sprachmodells operiert antikausal – es hat keine Vorstellung von Geschichte, sondern nur von statistischen Verteilungen. So muss es allein als formale Entität verstanden werden, die sich materialistischen oder historisierenden Lesarten entzieht – also auch jenem „forensischen Materialismus“, den der Autor Kirschenbaum selbst früher formuliert hat. Der Output eines solchen KI-Modells – wobei Kirschenbaum exemplarisch Ross Goodwins digitales Kunstwerk *I the Road* mit dem ganzen Arsenal der Literaturwissenschaft untersucht – erlaubt dann auch kein *close reading* mehr, sondern höchstens noch einen „reduktionistischen Formalismus“, der Formen, Muster und Entitäten im Text identifiziert, ohne deren tiefere Bedeutungen zu suchen.

Dem Transformer als Nachfolger des RNN widmet sich Hannes Bajohrs Beitrag. Er argumentiert sowohl gegen den auf die „großen“ Begriffe wie Intentionalität, Bewusstsein und Intelligenz abzielenden Hype der Tech-Giganten wie auch gegen die Behauptung, Bedeutung sei in KI grundsätzlich nicht zu haben.

Stattdessen spricht er sich, implizit auch gegen Kirschenbaum, für eine Art begrenzter, „dummer“ Bedeutung aus. In neuronalen Netzen, zumal dem Transformer, scheint eine artifizielle Semantik möglich zu sein, auch wenn sie noch weit von einem umfassenden Verständnis von Bedeutung entfernt ist. Bajohr untersucht sie anhand von zwei Beispielen: dem impliziten Wissen großer LLMs und der indexikalischen Bedeutung multimodaler KI wie Dall·E. Es ist diese Eigenschaft der Systeme, die den *humanities*, als Wissenschaften der Bedeutung, ein Einfallstor öffnet und „promptologische Untersuchungen“ ermöglicht.

Aber auch KI-Modelle „haben“ nicht einfach Sprache, sei ihre Bedeutung nun dumm oder schlicht abwesend. Sie werden mit großen Textmengen trainiert, die erst einmal ein *preprocessing* durchlaufen müssen. Tyler Shoemaker schlüsselt in seinem Beitrag genau auf, wie die sogenannte *subword tokenization* funktioniert, die die Masse an Inputtext nicht auf Buchstaben, sondern auf sogenannte Subwörter zurückführt und so komprimiert erst vektorisierbar macht. Diese basalen Einheiten, und nicht allein die ASCII- oder Unicode-Buchstaben, aus denen sie sich zusammensetzen, sind die Bausteine moderner digitaler, nämlich verketteter Textualität.

Wurden so Funktion und Aufbau großer Sprachmodelle diskutiert, wendet sich Leah Henrickson schließlich ihrer praktischen Anwendung in einem exemplarischen, gespenstischen Fall zu: dem Wiederauflebenlassen der Toten. In ihrer Untersuchung sogenannter „Thanabots“ – LLMs, die auf Texte Verstorbener trainiert wurden, um auch nach ihrem Ableben noch ein Gespräch mit ihnen zu ermöglichen – zeigt Henrickson, wie rhetorische, praktische und affektive Dimensionen in solchen Bots Eingang finden. Vertrauten Messaging-Apps ähnlich, erzeugen sie ein Gefühl der Kontrolle und parasozialer Bindungen. Zugleich zeigt Henrickson, wie diese Modelle hermeneutisch erschlossen werden können.

Der Band schließt mit einem Text, dem man heute bereits den Status eines historischen Dokuments zugestehen muss, auch wenn er erst 2021 erschienen ist. Das Paper prägte nicht nur den Begriff von LLMs als „stochastischen Papageien“, warnte vor den sozialen Effekten und Klimafolgen ihres Einsatzes und wandte sich gegen die *bigger is better*-Rhetorik ihrer aktuellen Entwicklung; infolge seiner Veröffentlichung wurden auch zwei der Autorinnen und damaligen Google-Angestellten Timnit Gebru und „Shmargaret Shmitchell“ (ein leicht zu durchschauendes Pseudonym) von ihrem Arbeitgeber freigestellt, der sich ungerecht kritisiert sah. Auch heute, wo BART oder GPT-3 längst nicht mehr den *state of the art* darstellen, ist es als konzentrierter Überblick über die Gefahren großer Sprachmodelle ein lesenswerter, wenngleich durchaus technischer Essay.

Dieser Band geht in Teilen auf einen Workshop im Frühling 2022 zurück, der am Seminar für Medienwissenschaft an der Universität Basel stattfand. Giulia Romani sei an dieser Stelle sehr herzlich für die Mitorganisation sowie die sorgfältige Durchsicht und Aufbereitung des Typoskriptes gedankt. Mögen die hier vorliegenden Beiträge jeder auf seine Weise, sei es inkrementell oder umstürzlerisch, einzeln oder *ensemble*, zu einer Stärkung der digitalen Kompetenz in den philologischen Disziplinen und darüber hinaus zu einem erhöhten Verständnis von algorithmischen Strukturen und zu einer kritischen Lesart von Texten in ihrer poetischen wie poetischen Handlungsmacht beitragen.

Die Herausgeber, Basel im Januar 2024

I. DAS SEQUENZIELLE PARADIGMA

CRITICAL CODE STUDIES. EIN MANIFEST

Mark C. Marino

1. Hello, World

„Hello, World“ ist eines der ersten Programme, das Lernende üblicherweise in einer ihnen neuen Programmiersprache schreiben. Dieses Programm, in der Regel nur ein paar Zeilen Code, veranlasst den Computer, eine Begrüßung auszugeben – ganz so, als ob er zu uns spräche. In LISP (*list-processing language*) sieht ein solches Programm zum Beispiel folgendermaßen aus:

```
(DEFUN HELLO-WORLD ()  
  (PRINT (LIST 'HELLO 'WORLD)))
```

DEFUN definiert die Funktion HELLO-WORLD (die hier ohne Argumente bleibt). Das Ergebnis: Der Computer bringt seinen Gruß an die Welt in unserer natürlichen Sprache vor. Welche Programmiersprache könnte für diese Aufgabe besser geeignet sein als LISP, eine Familie von algebraischen Listenverarbeitungssprachen, die für die Forschung mit künstlicher Intelligenz entwickelt wurden?¹ In diesem einfachen Programm erwacht der Computer in natürlicher Sprache zum Leben und spricht zu uns.

Aber natürlich versteht der Computer nicht, was er sagt. Ganz wörtlich genommen interpretiert der Computer diesen Code nicht einmal. Wenn die Funktion aufgerufen wird, gibt der Computer die Liste der beiden ‚Atome‘ (wie symbolische Einheiten in LISP genannt werden) *Hallo* und *Welt* aus. Die

¹ John McCarthy, „History of LISP“, in: *ACM SIGPLAN Notices* 13:8 (1978), S. 217–223.

einfachen Anführungszeichen im Skript weisen den Computer an, die Wörter *Hello* und *World* nicht zu interpretieren (ähnlich wie es die Kursivierung in diesem Satz tut). Mit dieser Unterscheidung wird Sprache aufgeteilt: in operativen Code auf der einen und Daten auf der anderen Seite. Der Computer mischt die Wörter hier lediglich, wie jede andere Folge von Daten, ohne sich um den Inhalt zu kümmern. Er interpretiert die Zeichenketten also nicht, er verarbeitet sie nur. Für uns – die Menschen, die den Code lesen – sind die Wörter in Anführungszeichen jedoch nicht sinnlos. *Hallo* und *Welt* haben eine Bedeutung, genauso wie der Funktionsname *PRINT* eine Bedeutung hat, die weit über seine Anweisungsfunktion an den Computer hinausgeht und in Richtung einer materiellen Kultur von Tinte und Schreiboberflächen weist. Wichtiger noch: Da das Wort *PRINT*, dieses ganz besondere Konstrukt, jede beliebige Zeichenfolge der Programmiersprache hätte sein können, offenbart die Entscheidung der Sprachdesigner:innen, gerade *PRINT* für diese Funktion zu wählen, einen zentralen Gedanken: Code existiert nicht für Maschinen, sondern für Menschen, die mit der Maschine und mit anderen Menschen kommunizieren müssen.

Und dennoch scheint es, als läge bis jetzt in der Forschung zur Computerprogrammierung der Code nur mit vorangestellten einfachen Anführungszeichen vor uns. Wir untersuchen zwar Programmierarchitekturen und bewundern Modularität und Effizienz, aber bis heute geht es der Untersuchung von Code nicht um Interpretation, um die Suche nach und die Produktion von Bedeutung. Das ist der Bereich der Geisteswissenschaften, der *artes liberales*, und nicht der Informatik. Selbst wenn die Ästhetik ihr Recht einfordert, geschieht dies als Aufruf zu stilistischer Klarheit, effizienterer Modifizierung und Entwicklung, obwohl die subjektiven Vorstellungen von ‚Schönheit‘ und ‚Eleganz‘ im Code eine Gelegenheit für einen noch zu führenden Dialog zwischen den Disziplinen böten.

Doch es gibt so viel mehr zu interpretieren. Über die Ästhetik des Codes hinaus sind da ideologische und diskursive Fragen (Konnotationen, Implikationen, Resonanz) sowie Aspekte von materieller und Sozialgeschichte, um nur einige der Untersuchungsbereiche zu nennen, die die interpretativen Wissenschaften zu bestellen suchen. Die Betonung der Funktionalität vernachlässigt damit verglichen die Bedeutung, die Code für sein menschliches Publikum hat. Denn Code, insbesondere im Fall von mittleren bis höherkomplexen Sprachen, existiert nicht nur für Computer – die mit Maschinensprache, also im Grunde der Repräsentation elektronischer Signale, oder gar mit elektronischen Signalen selbst arbeiten können –, sondern auch für Programmierer:innen. Der Computer kann also *ein* Empfänger des Codes sein – aber dann ist da noch die Person, die ihn schreibt, andere Programmierer:innen, Manager:innen und manchmal sogar Benutzer:innen, die Zugang zu seinem Quelltext haben. Tatsächlich hat sich heute das Publikum von Codes vergrößert, da nun Richter:innen und Anwält:innen, Politiker:innen und und sogar Dichter:innen Codes in ihre Diskurse einbringen.

Wenn wir also die Kritik von Code nur auf seine Funktionalität oder Ästhetik beziehen, dann nähern wir ihm uns lediglich mit einem kleinen Teil unserer analytischen Werkzeuge. In *Life on the Screen* beschreibt Sherry Turkle den *Julia-Effekt*, durch den Interagierende dem Computer – der vielleicht „Hello, World“ anzeigt – Intentionalität und Empfindungsvermögen zuschreiben.² Das bedeutet, auf Computer Menschlichkeit zu projizieren. Aber ist es nicht ebenso möglich, dass wir, wenn es um Code geht, genau das Gegenteil tun: den Code geradezu seiner menschlichen Bedeutung berauben, einfach weil wir der Meinung sind, dass er ein Zeichensystem darstellt, in dem die

² Sherry Turkle, *Life on the Screen: Identity in the Age of the Internet*, New York: Simon & Schuster 1997, S. 101.

umfangreichen Analysen semiotischer Systeme und Bedeutungen, von Konnotationen, die zu Denotationen werden, nicht gelten? Ist „Hello, World“ – ein Einführungsritual in viele Computersprachen – der Beginn einer Alphabetisierung, die durch seine eingeschränkte Interpretation selbst eingeschränkt bleiben muss? Was geschähe, wenn wir endlich begännen, die Bedeutung von Codes zu interpretieren?

Betrachten wir noch einmal das Beispiel des „Hello, World“-Programms in LISP. Ich habe bereits die Beziehung zwischen Programmiersprache und Code erwähnt, aber das ist nur der Anfang. Es hat sich herausgestellt, dass Programmierer:innen, die das Programmieren in LISP erlernen, viel eher mit einem Code beginnen, der die Fibonacci-Folge erzeugt, als mit „Hello, World“.³ Das Drucken oder Anzeigen von „Hello, World“ bietet nicht immer die direkteste Einführung in die Möglichkeiten einer Sprache. Ein Beispiel ist Ralph Westfalls alternatives „Hello, World“-Modell,⁴ das die strenge Typisierung und die objektorientierten Merkmale von Java hervorhebt – ein Ansatz also, der auf Alternativen setzt. In der Tat bietet die Debatte über die Verwendung von „Hello, World“ als Einführungsübung einen Einblick in die Art und Weise, wie das Wie und Warum selbst einfacher Programme zum Gegenstand von Diskursen und Analysen im sozialen Leben des Quellcodes werden kann.⁵

³ Richard Hollander vermutet, dass das mit der Rekursion von Sprache zu tun hat, die hier demonstriert wird: „Jede Sprache hat etwas, wofür sie gut ist“ (persönliches Interview, 3. März 2005).

⁴ Ralph Westfall, „Technical Opinion: Hello, World Considered Harmful“, in: *Communications of the ACM* 44:10 (2001), S. 129–130.

⁵ Es ist erwähnenswert, dass „Hello World“ seit der Veröffentlichung der ursprünglichen Version dieses Aufsatzes im *Electronic Book Review* zum „Hello, World“ der Critical Code Studies geworden ist, und in ganz verschiedenen Versionen interpretiert worden ist, zum Beispiel in James J. Brown Jr., *Ethical Programs: Hospitality and the Rhetorics of Software*, Ann Arbor: University of Michigan Press 2015; Annette Vee, *Coding Literacy: How Computer Programming Is Changing Writing*, Cambridge, MA: MIT Press 2017, und Matthew Kirschenbaum, „Hello Worlds“, in: *Chronicle of Higher Edu-*

Als Medienwissenschaftler mit literaturwissenschaftlichem Hintergrund möchte ich vorschlagen, nicht mehr nur metaphorisch vom Code als Text zu sprechen, sondern Code tatsächlich wie Text zu analysieren und zu erklären – als Zeichensystem mit eigener Rhetorik, als semiotische Kommunikation mit einer über ihren funktionalen Nutzen hinausgehende Bedeutung. Informatiker:innen machen sich Gedanken über die sinnvollste Verwendung von Code; die Geisteswissenschaften können ihren Teil beitragen, indem sie Vermutungen über die Bedeutung von Code für all jene anstellen, die ihm begegnen – sowohl direkt, durch unmittelbare Lektüre, als auch indirekt, durch die Auswirkungen der Programme, die dieser Code erstellt. Was ich in diesem Text also vorschlagen will, ist, dass wir Code so lesen und erklären, wie wir ein literarisches Werk oder andere Texte erklären. Das Feld, das ich damit eröffnen will, nenne ich Critical Code Studies (CCS).

Die Critical Code Studies sind ein Ansatz zur Erforschung von Code, der die kritische Hermeneutik auf die Interpretation von Computercodes, Programmarchitekturen und Dokumentationen in ihrem soziohistorischen Kontext anwendet. Die CCS gehen davon aus, dass die Codezeilen eines Programms nicht wertneutral sind und dass sie – zusätzlich zu den jeweiligen Interpretationsmethoden, die speziell für die Diskussion von Programmen entwickelt wurden – auch mittels jener theoretischen Ansätze analysiert werden können, die bei semiotischen Systemen Anwendung finden. Die CCS schließen insofern an die Arbeiten anderer *critical studies* an – etwa den *critical legal studies*⁶ und den *critical race studies*⁷ – insofern sie

cation, Januar 2009, <http://www.chronicle.com/article/Hello-Worlds/5476> [Letzter Zugriff: 17.7.2023].

⁶ Peter Fitzpatrick und Alan Hunt, „Introduction“, in: *Journal of Law and Society* 14:1 (1987), S. 1–3; Mark Tushnet, „Critical Legal Studies. A Political History“, in: *The Yale Law Journal* 100:5 (1991), S. 1515–1544.

⁷ Richard Delgado und Jean Stefancic, *Critical Race Theory. An Introduction*, New York: New York University Press 2001; Cornell West, „Foreword“, in: Kimberlé Crenshaw

kritische Theorie⁸ auf ein funktionales Dokument (ein juristisches Dokument oder Computerprogramm) anwenden, um eine Bedeutung zu explizieren, die über dessen Funktionalität hinausgeht, und behaupten, dass diese Bedeutung eine Analyse auf der Grundlage von mehr als einer Ästhetik der Effizienz rechtfertigt. Bedeutung erwächst aus der Funktionsweise des Codes, ist aber nicht nur auf die buchstäblichen Prozesse beschränkt, die der Code ausführt. Durch CCS können Praktiker:innen menschliche und datenverarbeitende Systeme als Ganze kritisieren – vom Computer bis zur Gesellschaft, in der diese Codeobjekte zirkulieren und ihren Einfluss ausüben.

Anstatt eine von der Arbeit der Programmierer:innen getrennte Sprache zu schaffen, werden die Critical Code Studies auf den bereits bestehenden Terminologien und Analysen aufbauen, die in der Coding-Community verwendet werden. Ein Großteil derzeitiger Codestudien scheint sich um die Effizienz, Wiederverwendbarkeit und Modularität von Code zu drehen. Der vorgeschlagene neue kritische Ansatz soll dagegen Bedeutung, Implikation und Konnotation betonen – allerdings nicht im Sinne eines in sich geschlossenen Bedeutungssystems, sondern im Hinblick auf dessen breiteren sozialen Kontext. Während ein:e Informatiker:in Gründe für oder gegen verschiedene pragmatische Ansätze vorbringen könnte, werden CCS-Forscher:innen die Kontexte und Implikationen hinterfragen, die den Entscheidungen von Programmierer:innen zugrunde liegen. Während sich ein:e Informatiker:in in erster Linie darauf konzentriert, wie der Code funktioniert oder wie er begrenzte Ressourcen nutzt, analysieren Vertreter:innen der Critical Code Studies die extrafunktionale Bedeutung des

u.a. (Hg.), *Critical Race Theory. The Key Writings That Formed the Movement*, New York: New Press, S. xi–xii.

⁸ *Critical theory* meint im Englischen nicht allein die Frankfurter Schule, sondern mithin die Gesamtheit kultur- und texttheoretischer Ansätze, die heute an amerikanischen Literaturdepartments zu finden sind. [Anm. d. Hg.]

Codes. ‚Extra‘ bedeutet hier nicht ‚außerhalb‘ oder ‚zusätzlich zu‘, sondern ‚aus dem Code hervorgehend‘. Die Bedeutung von Code ist mehrdeutig, da sozial, auch wenn er eindeutig, da technisch ist. Das Fachwissen von Programmierenden stammt nicht nur aus rein wissenschaftlicher Erkenntnis, sondern auch aus der Erfahrung mit der Kultur des Programmierens.

Obwohl einige der an den CCS Interessierten über fundierte Programmierkenntnisse verfügen, werden Analysen immer von der Zusammenarbeit mit kritischen Theoretiker:innen und Programmierer:innen profitieren. Genauso wie die Entwicklung neuer Medientechnologien nur durch die Zusammenarbeit von Künstler:innen und Programmierer:innen möglich ist, setzt auch die hier geforderte Forschung auf die virtuose Verbindung von Kenntnissen über Programmiersprachen und -methoden mit dem Fundus interdisziplinärer Theorien und Zugänge. Derartige analytische Projekte machen es notwendig, dass Programmierer:innen dabei helfen, den Inhalt und die Funktionsweise von Programmen zu erschließen, indem sie als Theoretiker:innen zusammen mit anderen Forscher:innen über eine Reihe von Beziehungen nachdenken. Diese Beziehungen bestehen zwischen, auf der einen Seite, dem Code selbst, seiner Architektur, seiner Funktionsweise und den spezifischen Programmierentscheidungen und -ausdrücken – und, auf der anderen Seite, dem, was der Code schließlich bearbeitet, ausgibt, prozessiert und darstellt.

Wie die Literaturanalyse sind die CCS ein interpretativer und kein instrumentell prospektiver oder nur deskriptiver Prozess. Andere Zweige (oder Fluchtlinien) unter den Codestudien mögen sich mit Pragmatik befassen, aber die CCS konzentrieren sich auf die Bedeutung, die aus der oft kollaborativen und sicherlich iterativen Leistung des Codens herausgelesen wird. Spezifische Ideologien des Codings, etwa die Open-Source-Programmierung, werden ebenfalls wichtig sein, obwohl die CCS davon ausgehen, dass Sprache, wie Gunther Kress und

Robert Hodge meinen, immer bereits selbst Ideologie ist.⁹ Wenn so auch „Software Ideologie ist“, wie Wendy Hui Kyong Chun mehrfach betont hat,¹⁰ dann ist es plausibel zu behaupten, Computercode sei ebenfalls Ideologie – aber eine Ideologie, die durch unser Analphabetentum und durch die Bildschirme, auf denen ihre Outputs erscheinen, doppelt versteckt ist.¹¹ Während Open Source also eine bestimmte Ideologie verkörpert, die den Code für die Analyse zugänglich macht, werden die CCS mehr als nur Open-Source-Programme analysieren, auch wenn der Zugang zu geschlossener Software begrenzter sein mag. Die CCS werden die Wahl der Paradigmen (z.B. objektorientierte Ansätze) im Großen, aber auch spezifische Codezeilen, ihre Einschränkungen und ihre Auswirkungen im Kleinen untersuchen. Nur durch genaue Beachtung der spezifischen Details des Codes wird CCS in der Lage sein, das zu artikulieren, was wir die *Konnotation* des Codes nennen könnten.

Die Critical Code Studies stützen sich auf die Arbeit anderer, die vor uns begonnen haben, Code zu interpretieren. Zu nennen ist insbesondere Matthew Fullers *Software Studies. A Lexicon*.¹² Fullers Sammlung, auf die weiter unten näher eingegangen wird, bietet ein Lexikon für die Interpretation von Software und ihrer Rolle in der Gesellschaft. Wenn *Software Studies* „ein Handbuch mit Ergänzungen zu einigen der wichtigsten Standardobjekte der Informatik, der Programmierung und der Software-Kultur“ ist,¹³ so benennen die Critical Code Studies eine Reihe von Ansätzen oder eine interpretative Haltung gegenüber Code. Tatsächlich passt Fullers Verständnis von Software

⁹ Gunther R. Kress und Robert Hodge, *Language as Ideology*, London: Routledge & Kegan Paul 1979.

¹⁰ Wendy Chun, *Sexuality in the Age of Fiber Optics*, Dissertation, Princeton University 1999, S. 207.

¹¹ Ebd.

¹² Matthew Fuller, *Software Studies: A Lexicon*, Cambridge, MA: MIT Press 2008.

¹³ Ebd., S. 2.

gut zu den Perspektiven der CCS, insofern er meint, dass „Software als Studienobjekt und Praxisbereich für Denkweisen und Arbeitsbereiche angesehen werden kann, die historisch betrachtet keine Software ‚besaßen‘ oder tatsächlich oft nicht viel Nützliches über sie zu sagen hatten“.¹⁴ *Software Studies* öffnet einer formalisierten Praxis der Critical Code Studies (und für eine ganze Reihe von Werkzeugen für die Beschäftigung mit ihnen) Tür und Tor.

Die Erwähnung von Literatur mag an dieser Stelle irreführend sein: Ich möchte die Critical Code Studies nicht auf die Untersuchung von Code beschränken, der *als* Literatur geschrieben wurde (wie das etwas bei der *Perl poetry* der Fall ist), obwohl diese Interpretation sicherlich nicht unberechtigt ist. Der Schwerpunkt der CCS liegt nicht auf der Herstellung von Code, der einen ästhetischen Wert und eine zusätzliche Bedeutung hat, sondern auf einer Sichtweise, die Code als eine Form des symbolischen Ausdrucks und der Interaktion betrachtet, die bereits eine Bedeutung jenseits seiner Funktionalität besitzt. Nichtsdestotrotz führen auch die Analysen von *Perl poetry* und von *codeworks* bereits eine Form der CCS durch und modellieren so Methoden zur Interpretation von Codezeilen, die auf andere Arten von Quellcode angewendet werden können.

Ich möchte CCS auch nicht mit dem „Literate Programming“ verwechseln, wie es von Donald Knuth, dem Autor von *The Art of Computer Programming*, beschrieben wurde.¹⁵ In *Literate Programming* (1984) schrieb Knuth: „Anstatt uns vorzustellen, dass unsere Hauptaufgabe darin besteht, einem Computer zu sagen, was er tun soll, sollten wir uns lieber darauf konzentrieren, den Menschen zu erklären, was der Computer tun soll.“ Knuth fährt fort: „Wer *literate programming* praktiziert, ist wie eine Per-

¹⁴ Ebd.

¹⁵ Donald E. Knuth, *The Art of Computer Programming*, Reading, MA: Addison-Wesley 1973.

son, die Essays schreibt – sie ist am ehesten darauf bedacht, sich klar auszudrücken und einen ausgezeichneten Stil zu pflegen.“¹⁶ Indem er Code als ein Mittel zur Kommunikation nicht nur mit Maschinen, sondern auch mit anderen Menschen identifiziert, kontextualisiert Knuth den Code als eine Art von Diskurs und betont die Stellung von Klarheit und Stilsicherheit als Maß seiner Lesbarkeit – seine Fähigkeit, den in diesem Code niedergelegten Zweck zu kommunizieren. Das Ziel der Critical Code Studies besteht jedoch nicht darin, Programmierer:innen dabei zu helfen, lesbareren Code zu schreiben, auch wenn das ein indirekter Effekt sein kann, sondern vielmehr darin, reichhaltige Methoden zur Lektüre dieses Codes zu entwickeln. Indem wir uns auf Knuths Arbeit stützen, können wir den vielen Aspekten des Codes, die wir analysieren, Stil und Klarheit verleihen – aber dabei dürfen wir nicht stehen bleiben.

Grundlegend für die CCS ist die Annahme, dass Code ein soziales, semiotisches System ist, das Grammatik und Rhetorik mobilisiert. Rita Raley schreibt: „Code mag in einem allgemeinen Sinne undurchsichtig und nur für Spezialist:innen lesbar sein, ähnlich wie das Zeichensystem einer Höhlenmalerei – aber er ist eingeschrieben, programmiert, niedergelegt worden. Er ist konkret historisch bedingt.“¹⁷ Und Adrian Mackenzie merkt an: „Code wird in situieren Praktiken geschrieben und ausgeführt, in Bezug auf bestimmte Bereiche und innerhalb bestimmter Ordnungen und Unordnungen des kollektiven Lebens. Seine Formen und Abstraktionen sind mit dem Leben verbunden.“¹⁸ In der Art und Weise, wie dieses Zeichensystem

¹⁶ Ebd., S. 97.

¹⁷ Rita Raley, „Code.Surface || Code.Depth“, in: *Dichtung Digital* 8.1 (2006), <http://dx.doi.org/10.25969/mediarep/17695> [Letzter Zugriff: 17.7.2023].

¹⁸ Adrian Mackenzie, „The Problem of Computer Code: Leviathan or Common Power“, Institute for Cultural Research Lancaster University, März 2003, https://www.academia.edu/2718446/The_problem_of_computer_code_Leviathan_or_common_power [Letzter Zugriff: 17.7.2023], hier S. 19.

in den Akteursnetzwerken von Computern und Maschinen zirkuliert, entwickelt es interpretationswürdige Konnotationen.

Friedrich Kittler warnt in seinem Eintrag in *Software Studies* unter dem Titel „Code oder wie sich etwas anders schreiben lässt“ vor einer Verwässerung und damit Mystifizierung und Vereinheitlichung des Begriffs ‚Code‘.¹⁹ (Kittlers eigenem Code ist Kapitel 6 meines Buchs *Critical Code Studies* gewidmet.)²⁰ Kittler warnt: „Codes sind nach Wort und Sache, was uns heutzutage bestimmt und was wir daher sagen müssen, schon um nicht unter ihnen zu verschwinden. [...] Der Begriff des Codes, heißt das aber, ist so inflationär wie fraglich.“²¹ Kittler, der sich einer losen, assoziativen Deutung widersetzt, führt die Herleitung des Wortes auf *codex* und weiter auf *codicilla* zurück, „kleine geschälte Blöcke aus Holz, in deren Wachsbeschichtung sich schreiben ließ“. In seiner späteren Form, dem *Codex*, bedeutet das Wort „schlicht das gebundene Gesetzbuch“. Der Code wird so zum Mittel und Medium der Fernsteuerung. Kittler verfolgt den Begriff von den klassischen Imperien über die Nationalstaaten bis zu dem Moment, wo er zum Synonym für *Chiffre* wird.²²

Doch Code kritisch zu interpretieren ist keinesfalls dasselbe, wie ihn zu dechiffrieren. Schon der Begriff der ‚Interpretation‘ kann ein Hindernis auf der Brücke zwischen Informatik und Geisteswissenschaften darstellen. Ein eher mathematisch denkender Mensch würde vielleicht die Verwendung von Isomorphismen dieser ungenauen Beziehung vorziehen. Douglas Hofstadter versteht in *Gödel, Escher, Bach. Ein Endloses Geflochtenes Band* unter ‚Interpretation‘ sinnvolle Isomorphismen, also

¹⁹ Der Beitrag aus *Software Studies* ist eine Übersetzung, die hier im Original zitiert wird. [Anm. d. Hg.] Friedrich Kittler, „Code oder wie sich etwas anders schreiben lässt“, in: Gerfried Stocker und Christine Schöpf (Hg.), *Code. The Language of our Time*, Linz: Hatje Cantz 2003, S. 15–19.

²⁰ Mark C. Marino, *Critical Code Studies*, Cambridge, MA: MIT Press 2020.

²¹ Kittler, „Code“, hier S. 15 und 18.

²² Ebd., S. 16.

Beziehungen zwischen einem System und einem anderen. In Hofstadters Formulierung ist ein Isomorphismus oder eine Interpretation nur dann sinnvoll, wenn die Wahrheiten in einem System auch Wahrheiten in einem anderen System schaffen. Um einen Isomorphismus zu erzeugen, müssen die beiden Systeme vollständig austauschbar sein, sodass jedes Zeichen in dem einen System mit einem Zeichen in dem anderen System gleichgesetzt werden kann, was im Wesentlichen eine Dechiffrierung oder Decodierung darstellt.²³ Diese Form der Interpretation ähnelt viel mehr der Transliteration oder gar dem Umkompilieren von einer höheren in eine niedrigere Programmiersprache als der Interpretation im geisteswissenschaftlichen Sinne. Im Gegensatz dazu geht es bei Letzterer weniger darum, eine Eins-zu-eins-Korrespondenz abzubilden, sondern Verbindungen und Resonanzen zwischen Zeichen und Referenten zu erkennen sowie Diskontinuitäten und Abweichungen zusammen mit jenen Kräften auszumachen, die Bedeutung herstellen oder verzerren.

Das PRINT aus unserem „Hello, World“-Programm ist zum Beispiel ein Token, mit dem bewirkt wird, dass ebenjene Daten auf dem Bildschirm angezeigt werden. Wie wir im Buch *10 PRINT* diskutieren, hat der Ausdruck PRINT seinen Ursprung in einem Befehl, der eigentlich Tinte auf Papier bringen soll. Man kann also eine Isomorphie ziehen zwischen dem Befehl PRINT als Befehl zur Anzeige von Text auf dem Bildschirm und dem Fernschreiber, der ein Wort aufs Papier bringt; oder zwischen PRINT und einem ähnlichen Token in einem anderen Programm Sprache, wie z.B. `writeLine` in C#. Der Befehl PRINT kann jedoch auf noch viel assoziativere Arten interpretiert werden. *Print* erinnert an ein Aufschreibesystem, an die Geschichte des Druckwesens, an die Art und Weise, wie der Computer-

²³ Douglas R. Hofstadter, *Gödel, Escher, Bach. Ein Endloses Geflochtenes Band*, München: dtv 2008, S. 54.

bildschirm auf Papier gedruckten Text remediiert.²⁴ Obwohl ein Isomorphismus Parallelen bietet, kann Interpretation auf sehr viel weiter gehende Bedeutungsbereiche hinweisen.

Interpretation ist nicht auf die Kompilierung oder die Portierung von Code beschränkt. Zwar ähnelt Interpretation im geisteswissenschaftlichen Sinne eher der Portierung eines Programms von einer Sprache in eine andere, bei der ein:e Programmierer:in bewusste Entscheidungen darüber treffen muss, wie ein System in einem anderen dargestellt werden soll, und die Portierung eine subjektive Beurteilung dessen erfordert, was das Wesen des Programms ausmacht. Aber die geisteswissenschaftliche Interpretation geht über Kompilieren und Portieren hinaus, indem sie fragt: „Wenn x für y steht, was wird dann über beide Systeme kommuniziert?“ Sie will wissen: „Wie informiert oder beeinflusst ein Bedeutungsbereich einen anderen?“ Mit anderen Worten: Interpretation im geisteswissenschaftlichen Sinne ist weder Dechiffrierung noch Übersetzung; sie nutzt diese Techniken (und andere hermeneutische Verfahren), um Bedeutung jenseits der isomorphen Systeme zu erfassen. Interpretation ist daher nicht nur Decodierung, sondern die Erzeugung einer *anderen* Art von Wissen: Einsicht. Diese Einsichten können freilich recht willkürlich erscheinen, wenn man sich dem Code mit einem systematischen Blick nähert – wenn man sich also an die Logik der Systeme hält, anstatt sie potenziell infrage zu stellen.

Kittler bietet dieses robustere Interpretationsmodell auf, wenn er die Ursprünge und Kontexte von Computercode untersucht. In seiner historischen Erzählung erweisen sich Codes als zentral für Techniken der Unterwerfung in dem Zeitraum, da sich das militärische Motto von „Command, Control, Communication, Intelligence“ (CCCI) zu „Command, Control,

²⁴ J. David Bolter und Richard Grusin, *Remediation. Understanding New Media*, Cambridge, MA: MIT Press 1999. Die Autoren bezeichnen mit dem Begriff ‚Remediation‘ den Prozess, durch den ein Medium in einem anderen neu geschaffen wird.

Communication, Computers“ (C⁴) ändert. „Technik heute setzt den Code in Wirklichkeiten um, codiert also die Welt.“²⁵ Zentral für derartige Kontrolle ist die Fähigkeit, den „Code“ zu einem allgegenwärtigen, universalen Operator zu machen. Und Kittler fügt hinzu: „Womöglich aber heißt Code, wie Codex ja einst auch nur das Gesetz genau des Imperiums, das uns unterworfen hält und sogar diesen Satz zu sagen untersagt. Mit triumphaler Gewissheit jedenfalls verkünden die Großforschungseinrichtungen, die am meisten davon profitieren, nichts sei im Weltall, was nicht Code sei, vom Virus bis zum Big Bang.“²⁶

Hier setzt der militärisch-industrielle Komplex Codes zur Kontrolle ein, und indem er den „Code“ zum Wesen allen Lebens macht, etablieren sich einige wenige elitäre, gebildete Forscher:innen als die einzig möglichen Vermittler:innen. Um die Kontrolle über den Begriff wiederzuerlangen, versucht Kittler, ihn in einer technischen Definition zu bändigen. Er verkündet: „Codes sollten daher einzig Alphabete im Wortsinn der modernen Mathematik heißen, eineindeutige und abzählbare, ja, möglichst kurze Folgen von Symbolen also, die dank einer Grammatik mit der unerhörten Fähigkeit begabt sind, sich gleichwohl selbst unendlich zu vermehren.“²⁷ Kittler grenzt den Begriff des Codes hier ein – aber zu welchem Zweck? Die Critical Code Studies mögen die von Kittler vorgeschlagene Zurückhaltung aufnehmen, um Code nicht zu einem alles beherrschenden, transzendenten Imaginären werden zu lassen. Diese eineindeutigen Systeme namens Codes entwickeln jedoch mehrdeutige Konnotationen in dem Moment, da sie in die soziale Sphäre eintreten – mit anderen Worten, wenn sie ausgesprochen oder auch nur gedacht werden, wenn sie im Kontext des menschlichen Bewusstseins, in bedeutungsgeben-

²⁵ Kittler, „Code“, S. 18.

²⁶ Ebd.

²⁷ Ebd.

den Maschinen entstehen. Das Ziel, Codes im Kontext zu untersuchen, lässt die CCS den Blick auf und über Zeichen hinaus richten, um Gesten, Performances und Möglichkeiten zu interpretieren.

2. Was ist interpretierbar?

Alles. Der Code, die Dokumentation, die Kommentare, die Strukturen, die kompilierten Versionen – alles steht der Interpretation offen. Ein besseres Verständnis dieser Elemente (und der Zugang zu ihnen) wird Praktizierenden der CCS helfen, zunehmend komplexe Lesarten zu entwickeln. In seinem Aufsatz „The Code Is Not the Text (Unless It Is the Text)“ argumentiert John Cayley, dass ein Großteil jener Codes, der in Werken der elektronischen Kunst analysiert wurde, gar kein Code im eigentlichen Sinne war, sondern lediglich dafür ausgegeben wurde.²⁸ Cayley meinte, dass Code nur dann „nicht eigentlich der Text ist“, wenn es sich um ausführbaren Code handele.²⁹ In „A Box, Darkly“ konterten Michael Mateas und Nick Montfort Cayleys Behauptung einer notwendigen Ausführbarkeit, indem sie bemerken, dass Code für Programme geschrieben werden kann, die überhaupt *nie* ausgeführt werden.³⁰ Im Rahmen der CCS aber trägt alles, was Teil des Programms oder seines Paratexts ist (im weitesten Sinne), zur Bedeutungsbildung bei. Wenn das Artefakt codeähnlich ist, kann es die Art und Weise, wie wir Code lesen, beeinflussen. Ich würde auch Interpretationen von

²⁸ John Cayley, „The Code Is Not the Text (Unless It Is the Text)“, in: *Electronic Book Review*, September 2002. <https://electronicbookreview.com/essay/the-code-is-not-the-text-unless-it-is-the-text/> [Letzter Zugriff: 17.7.2023].

²⁹ Schon bei der Präsentation auf der Modern Language Association, bei der ich dieses Manifest erstmals vorstellte, hatte Cayley diese Position gelockert.

³⁰ Michael Mateas und Nick Montfort, „A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics“, in: *Proceedings of the 6th Digital Arts and Culture Conference*, S. 144–153.

Markup-Sprachen und Skripten als Erweiterungen von Code betrachten. Innerhalb solcher Codes werden die Interpretierenden die eigentlichen Symbole untersuchen, aber auch im weiteren Sinne Verfahren, Strukturen und Gesten. Es geht um paradigmatische Entscheidungen, die bei der Konstruktion des Programms getroffen werden, um Methoden, die anderen vorgezogen werden, und um Konnotationen.

Zusätzlich zu den Symbolen und Zeichen in den Programmdateien selbst sind diese paratextuellen Merkmale für kompetente Leser:innen ebenfalls von einiger Wichtigkeit. Die Geschichte des Programms, der Autor oder die Autorin, die Programmiersprache, das Genre, die Finanzierungsquelle, die hinter der Entwicklung stand (sei sie militärisch, gewerblich oder die Unterhaltungsbranche) – all das hat einen Einfluss auf Bedeutung, auch wenn bei einer Lektüre vielleicht nur einige dieser Aspekte im Vordergrund stehen. Das Ziel muss nicht die Analyse von Code um des Codes willen sein; aber die Analyse von Code hat das Ziel, Programme besser zu verstehen sowie die Netzwerke, die zwischen Programmen und Menschen bestehen – die Interaktion, Organisation, Repräsentation, Manipulation und Transformation und alle anderweitigen Verflechtungen zwischen ihnen. Die Lektüre von Code dient so als Einstieg in die Lektüre von Kultur.

Code hat mehr als ein Publikum. Auf den ersten Blick scheint der einzige Kandidat für ein solches Publikum nur die Maschine selbst zu sein, aber es sind, wie erwähnt, allein Menschen, die eine derartige symbolische Darstellung benötigen. Erstens liest der:die Programmierer:in den Code selbst schon während des Schreibprozesses (vorausgesetzt, er wird von einem Menschen und nicht von einer anderen Software erstellt). Zweitens können andere Programmierende den Code lesen. Drittens können auch Nicht-Programmierer:innen, wie Projektmanager:innen oder das Publikum interdisziplinärer Konferenzen, den Code zu lesen bekommen. Nicht zuletzt kann

der Code unberechtigten Hacker:innen in die Hände gelangen. Selbst die Maschine, auf der der Computer läuft, kann sich als multiples Publikum erweisen, da Teile des Codes an andere Prozesse und andere Teile der Maschine weitergegeben werden. Mehr noch: Es gibt heute bereits eine Vielzahl von Orten, an denen über Code gesprochen wird. Jeremy Douglass ruft dazu auf, Code-Debatten „in freier Wildbahn“ zu finden, und verweist auf Diskussionen in Geschäftsbesprechungen oder vor Gericht und sogar auf die Abendnachrichten.³¹

Um auf das Beispiel zurückzukommen, mit dem ich dieses Kapitel begann: Das LISP-Programm „Hello, World“ ist nicht aus dem Nichts aufgetaucht. Es wurde von Mark Singletary geschrieben, einem Programmierarchitekten, der sowohl bei der NASA als auch bei Softwarefirmen beschäftigt war. Sein vordergründiges Spezialgebiet war *usability* – die Entwicklung leicht benutzbarer Mensch-Computer-Schnittstellen zur Erleichterung der Interaktion zwischen Mensch und Maschine. Die Beziehung zwischen Code und Coder:in – zwischen dem Programm, das Computer dazu bringt, mit einer „Welt“ zu sprechen, und den Programmierenden, die (für eine staatlich finanzierte weltraumforschende C⁴-Institution) daran arbeiten, Computer verständlicher sprechen zu lassen – schafft Bedeutung. Darüber hinaus reichte Singletary diesen „Anfänger:innen“-Code beim „Hello World“-Projekt der Association for Computing Machinery (ACM) ein,³² einem Projekt, das allein schon durch seine bloße Existenz zusätzliche Leser:innenschaften für den Übungscode nahelegt. Was bedeutet es für die ACM, Bei-

³¹ Jeremy Douglass, „Critical Code Studies Conference—Week Two Discussion“, in: *Electronic Book Review*, April 2011, <http://electronicbookreview.com/essay/critical-code-studies-conference-week-two-discussion/> [Letzter Zugriff: 17.7.2023].

³² Mark Singletary, „Hello World‘ Project“, <http://www2.latech.edu/~acm/helloworld/> [nur noch über die *Wayback Machine* aufzurufen: <https://web.archive.org/web/20001025203218/http://www.latech.edu/~acm/helloworld/HelloWorld.shtml>]. [Letzter Zugriff: 17.7.2023].

spiele für den „ersten Kontakt mit einer neuen Sprache“³³ zu sammeln? Was bedeutet es, wenn eine Übung, ein Initiationsritual, anthologisiert wird?

Plötzlich hat dieser Übungscode eine Bühne.³⁴ Damit hat sich die „Welt“, die das Programm begrüßt, auf andere Programmierende an Computern mit Internetzugang ausgeweitet (Bedingungen, die die Welt auf einen absurd kleinen Teil der Erdbevölkerung einschränken). Dieser spezielle Code wird Teil vergleichender Lektüre und nicht nur ein Beweis für angehenden Programmieralphabetismus. Außerdem ist er kaum die einzige Möglichkeit, ein „Hello, World“-Programm in LISP zu schreiben. Es bietet also, wie alle semiotischen Äußerungen, einen Einblick in das, was Saussure das „Paradigma“ möglicher Formulierungen nennt.³⁵ Ohne Frage ist dieses Programm nicht nur ein Stück Code, das vom Computer verarbeitet wird. Es ist ein Artefakt, das innerhalb eines breiteren Austauschs zirkuliert. Obwohl ich keine endgültigen Lesarten der materiellen, soziohistorischen Faktoren dieses Code-Objekts anbieten werde und damit riskiere, die CCS von vornherein unnötig einzuschränken, möchte ich lediglich aufzeigen, wie all diese kontextualisierenden Aspekte potenziell die Bedeutung von „Hello, World“ beeinflussen.

Der Schlüssel zu den Critical Code Studies ist die Programmierkompetenz. Genauso wie Studierende der Literatur eine Fremdsprache, eine Gebärdensprache oder einen Tanz erlernen können, werden Praktizierende der CCS nur in der Lage sein, auf den Code zuzugreifen, indem sie dessen Sprache lernen (und vorzugsweise auch hinreichend beherrschen).³⁶ Rita

³³ Ebd.

³⁴ Die Webseite wurde am 1. Februar 1996 sogar mit dem Preis „Geek Site of the Day“ ausgezeichnet, wie es auf der Webseite des Projekts heißt.

³⁵ Ferdinand de Saussure, *Grundfragen der allgemeinen Sprachwissenschaft*, Berlin: De Gruyter 2001, S. 147–151.

³⁶ Eine ausführliche Diskussion über Programmierung als Alphabetisierung findet sich in Vee, *Coding Literacy*.

Raley hat über die Möglichkeit nachgedacht, Computersprachen den Sprachanforderungen in Studiengängen der Geisteswissenschaften anzurechnen.³⁷ Alternativ könnte ein:e Interpret:in mit jemandem zusammenarbeiten, der die Sprache beherrscht, und so Kompetenznetze, Koalitionen der Bedeutungsbildung aufbauen. Man könnte eine solche Sprachkompetenz als *Cyborg-Alphabetismus* bezeichnen. Informatiker:innen schreiben bereits Übersetzungen in Form von Portierungen ihrer Programme. Vielleicht werden sie eines Tages sogar Anleitungen zu ihrer Exegese schreiben. In Lehrbüchern über Programmierung tun sie zweifellos etwas in dieser Art.

Auch wenn es den Anschein hat, dass Literaturwissenschaftler:innen lediglich die Elemente natürlicher Sprache interpretieren können, dürfen sich die Critical Code Studies nicht nur auf die natürliche Sprache in oder willkürliche Merkmale von Codes beschränken. Ein Programm zur Berechnung von Kfz-Versicherungstarifen könnte zum Beispiel eine Variable namens *victim* beinhalten. Variablen- und Methodennamen sind jedoch relativ leicht zu interpretieren und setzen sich dem Vorwurf aus, willkürliche Elemente des Codes zu sein, die für das Funktionieren der Maschine relativ bedeutungslos sind. Die Variable *victim* könnte genauso gut *f2s989* heißen. Einerseits ist die Beziehung zwischen den meisten Signifikanten und Signifikaten in semiotischer Hinsicht willkürlich und dennoch recht bedeutungsvoll. Andererseits scheinen die Namen der vorprogrammierten Methoden, die Syntax und Logik einer Computersprache, eine deutlichere operative Bedeutung für das Computersystem zu haben. Daher wären Critical Code Studies, die nur von Variablennamen und niemals von Sprach-Tokens und ihren Grundlagen sprechen, recht oberflächlich. Der ehemalige professionelle Programmierer und Technolo-

³⁷ Rita Raley, „Interferences: [Net.Writing] and the Practice of Codework“, in: *Electronic Book Review*, September 2002, <http://electronicbookreview.com/essay/interferences-net-writing-and-the-practice-of-codework/> [Letzter Zugriff: 17.7.2023].

giekulturwissenschaftler Evan Buswell hat diesen Punkt sehr treffend formuliert.³⁸ Sich die Folgen eines solchen Ansatzes ausmalend, sieht Buswell eine Zukunft voraus, in der sich eine Gruppe von Programmierer:innen darum kümmert, dass die Dinge funktionieren, während die Code-Kritiker:innen in einem Nebenraum über Variablen- und Methodennamen diskutieren.

Critical Code Studies sind nicht auf eine bestimmte Art von Programmierschema beschränkt, sei es imperativ, objektorientiert, funktional usw. Auch sollten sie nicht auf Programmiersprachen beschränkt sein, die textliche Symbole verwenden. Die CCS basieren auf semiotischer Analyse, sodass ihre Methoden auf jedes Repräsentationssystem angewendet werden können, sei es textbasiert, visuell oder sogar gestisch. Damit die CCS wirklich robust sind, sollten ihre Methoden auch auf die direkte Programmierung durch Hardware anwendbar sein – zum Beispiel beim Bau von Mikroschaltkreisen. Auch wenn einige der Meinung sein mögen, dass eine solche Analyse in die *platform studies* eingreift, wird ein allzu starres Beharren auf diese Grenzen nur die komplexen Wechselbeziehungen zwischen Software und Hardware bemänteln. Mit anderen Worten: Die CCS sollten ihre Analyse nicht einfach einstellen, wenn sie einen begrenzten Zuständigkeitsbereich überschreiten, sondern müssen stattdessen mit den verwandten Studienbereichen zusammenarbeiten, die ein umfassendes Verständnis der zeitgenössischen Technologiekultur ermöglichen.

Computercode ist kein unentdeckter Kontinent. Es handelt sich auch nicht um ein magisches Zeichensystem. Er ist eine Sphäre von Kreativität und Produktivität, die es seit mehr als einem halben Jahrhundert gibt. Es wäre ein schrecklicher Fehler, sich in Diskussionen über Code einzumischen, indem man

³⁸ Evan Buswell, Kommentar zu Max Feinstein u.a., „Critical Code Studies“, <https://www.hastac.org/initiatives/hastac-scholars/scholars-forums/critical-code-studies> [Letzter Zugriff: 17.7.2023].

sich anmaßt, ihn für die Programmierenden zu interpretieren, die sich schon so lange darin abmühen. Damit würde man den Tatbestand des Techno-Orientalismus erfüllen. Es gibt zahlreiche Definitionen für einen solchen Akt, aber sie alle gehen auf Edward Saids Bezeichnung für die Art und Weise zurück, wie Gelehrte im Westen dem undurchschaubaren Orient, der asiatischen Kultur und Sprache, einen Sinn zuweisen und dann dazu übergehen, ihn zu interpretieren, ohne seine inneren Zusammenhänge zu verstehen oder gar diejenigen einzubeziehen, die bereits einen Diskurs über seine Interpretation geführt haben. Es ist der Inbegriff einer kulturimperialistischen Hybris, sich anheischig zu machen, die Kultur einer:s anderen zu erklären. Die Critical Code Studies müssen daher in Zusammenarbeit und im Gespräch mit denen arbeiten, die bereits die Grundlagen für dieses analytische Bemühen gelegt haben.

3. Der Code als Mittel, nicht als Zweck

An dieser Stelle könnte jemand fragen: Sollten wir nicht viel eher Maschinensprache interpretieren? Vielleicht. Aber das würde uns nicht näher an die Wahrheit des Codes bringen. Darum geht es uns nicht. Solche Interpretationen könnten Vorbilder in der Kritik von Zahlen finden, wie etwa Brian Rotmans *Signifying Nothing* nahelegt, dass aus dem Nichts viel herauszulesen ist.³⁹ Gleichungen haben ihr eigenes Leben, wie Werke wie $E = mc^2$ von David Bodanis zeigen.⁴⁰ Einer der vielleicht aussagekräftigsten Texte, der den Übergang zwischen Darstellungsformen und Ausdruckssystemen veranschaulicht, ist Hofstadters *Gödel, Escher, Bach*, in dem Korrespondenzen zwischen

³⁹ Brian Rotman, *Signifying Nothing: The Semiotics of Zero*, Stanford, CA: Stanford University Press 1993.

⁴⁰ David Bodanis, *E = mc²: A Biography of the World's Most Famous Equation*, New York: Walker Books 2000.

natürlichen Sprachen, Mathematik, Programmierung, Musik und Zeichnung hergestellt werden.⁴¹ Wo die Grenze zwischen Mathematik und CCS verläuft, möchte ich nicht entscheiden, sondern lediglich darauf hinweisen, dass es Eigenschaften von Computersprachen gibt, die einem Programm mehr Spielraum für Interpretationen geben.

Es ist jedoch wichtig, nicht der Illusion zu verfallen, das Lesen von Assembler- oder Maschinensprache oder Binärcode sei gleichbedeutend mit dem Lesen eines wahren Codes. Wendy Chun hat sich besorgt über die Tendenz geäußert, den Quellcode als etwas Wesentliches zu betrachten, als das Herzstück des Programms, und ihn sogar mit dem ausführbaren Code zu verwechseln, in den der Quellcode kompiliert wird. Bei einer solchen Suche besteht die Gefahr, dass man sich auf eine Art „Sourcery“ einlässt.⁴² Die Frage ist nicht, ob es besser ist, eine höhere oder niedrigere Sprache zu lesen, sondern was jemand herausfinden möchte. Patrick LeMieux hat zum Beispiel die Assemblersprache von Cory Arcangels *Super Mario Clouds* einer Lektüre unterzogen, um die Mechanismen dieses Werks zu untersuchen.⁴³ Natürlich bietet der Maschinensprachcode Zugang zu einer Ebene, die direkter mit der Hardware verbunden ist.

Chuns Kritik ist nicht nur die Mahnung, dass der Quellcode ein Zwischenprodukt in der Darstellung maschineller Befehle ist; ihr Beitrag legt auch nahe, dass keine Ebene des Codes privilegiert werden darf. Ich möchte dem hinzufügen, dass Bedeutung erst entsteht, wenn der Code von menschlichen Leser:innen interpretiert wird. Obwohl also keine Ebene des Codes bedeu-

⁴¹ Hofstadter, *Gödel, Escher, Bach*.

⁴² Wendy Chun, *Programmed Visions: Software and Memory*, Cambridge, MA: MIT Press 2011.

⁴³ Patrick LeMieux, „Everything but the Clouds. Platform Games“, Babycastles, New York 2015, verfügbar unter: <https://vimeo.com/241966869> [Letzter Zugriff: 17.7.2023].

tungsvoller ist als eine andere, ist jede Ebene des Codes und der Hardware eine potenzielle Quelle von Bedeutung.

Obwohl Kritiker:innen nicht auf eine tiefere Ebene des Codes abtauchen müssen, um mehr Bedeutung ausfindig zu machen, müssen sie doch darauf achten, den Code nicht als selbstgenügsam zu verstehen. Matthew Kirschenbaum hat bei mehreren Gelegenheiten auf die Gefahren hingewiesen, die sich ergeben, wenn man den Code losgelöst von seinem Kontext liest.⁴⁴ Er warnt davor, beim Code stehen zu bleiben und dabei die vielen anderen Systeme aus den Augen zu verlieren, mit denen der Code interagiert und auf die er einwirkt. Stattdessen sollte ein:e Kritiker:in untersuchen, wie der Code mit anderen Systemen und Hardware interagiert. High-Level-Code ist über mehrere Plattformen hinweg portabel, aber er ist nicht weniger situiert. Auch wenn die jeweiligen Situationen, in denen Code steht, zu zahlreich sind, um sie aufzuzählen, sind seine Software- und Hardwarekontexte doch immer auch entscheidend für seine Interpretation.

Mit diesen beiden Warnungen im Hinterkopf konnten wir, die wir damit begonnen haben, Critical Code Studies zu betreiben, einige Praktiken entwickeln. Die erste dieser Praktiken ist die Anerkennung, dass der Code eher als Ausgangspunkt für eine Untersuchung dient, als ein Selbstzweck zu sein. Das vielleicht deutlichste Beispiel für diese Methode war die gleichnamige Studie über ein einzeliges Basic-Programm: `10 PRINT CHR$(205.5 + RND (1)); : GOTO 10`.⁴⁵ Die Untersuchung mit zehn Autor:innen nutzte dieses einfache, aber einflussreiche Programm als Ausgangspunkt für eine Diskussion über die Programmiersprache BASIC, den Commodore 64, prozedurale Kunst und sogar Labyrinth. Obwohl jedes Token des Pro-

⁴⁴ Matthew Kirschenbaum, „<!--Opening Thoughts-->“, <https://www.hastac.org/initiatives/hastac-scholars/scholars-forums/critical-code-studies> [Letzter Zugriff: 17.7.2023].

⁴⁵ Nick Montfort u.a., *10 PRINT CHR\$(205.5+RND(i)); : GOTO 10*, Cambridge, MA: MIT Press 2013.

gramms (10, PRINT, CHR\$ usw.) analysiert wurde, kamen doch auch die Begriffe Zufälligkeit und Regelmäßigkeit, Hardware und Spezifikationen sowie Designprinzipien, die das Programm ermöglichten, zur Sprache – materielle Aspekte, wie die Konfiguration des Commodore-64-Monitors, nicht weniger als soziale Aspekte, wie der Zeitgeist des „Home Computing“. Das Programm war kein Strandball, der in ungerichteter freier Assoziation hin und her gespielt wurde, sondern die Nabe eines Rads, dessen Speichen die Untersuchungsrichtungen vorgaben, die aber immer wieder zu ihrer Achse zurückwiesen. Und wie es sich für ein solches rekursives Programm gehört, führte der Prozess am Ende in sein Zentrum, zu seiner scheinbar unendlichen potenziellen Energie. Selbst die Analyse eines Programms, das nur eine Zeile lang ist, kann also nicht isoliert durchgeführt werden.

Eine weitere Entwicklung, die aus dieser Intervention hervorging, war das Bewusstsein für den Bedarf an Werkzeugen, mit denen Code diskutiert werden kann, ohne dass er aus seiner Umgebung herausgelöst werden muss. Als Lösung für die Herausforderungen der Kontextualisierung von Code schlug Kirschenbaum vor, Code innerhalb seiner Versionskontrollumgebung zu lesen. Wir haben begonnen, mit einer solchen Möglichkeit in Projekten zu experimentieren, die auf ANVC Scalar und einer speziellen Adaption davon namens ACLS Workbench beruht.⁴⁶ In beiden Kontexten ermöglicht die Scalar-Softwareplattform den Import von Code als Text aus Quellcode-Repositoryn, wobei die Verbindung zu seinem umfassenderen Kontext erhalten bleibt. Bei meiner Lektüre des *Transborder Immigrant Tool*⁴⁷ habe ich zum Beispiel Scalar ver-

⁴⁶ ACLS Workbench wurde vom American Council of Learned Societies finanziert und von Lucas Miller in Zusammenarbeit mit Craig Dietrich und Erik Loyer als eine Abspaltung von ANVC Scalar entwickelt.

⁴⁷ Mark C. Marino, *Critical Code Studies*, Cambridge, MA: MIT Press 2020, Kap. 3. [Anm. d. Hg.]

wendet, um bestimmte Dateien aus dem Repository des Codes zu importieren, sodass sie immer noch in ihrem Kontext standen, der das viel größere Stück Software ausmachte. Mit dieser Technik des Imports von Dateien in einem bestimmten Kontext können Kritiker:innen den Code an Ort und Stelle kommentieren, anstatt ihn zu extrahieren und aus seiner Entwicklungsumgebung zu entfernen. Dieser Ansatz beinhaltet immer noch eine Abstraktion, da der Code über einen Browser gelesen wird, der von der Plattform (oder sogar einem Emulator dieser Plattform) getrennt ist, auf der er laufen soll. Nichtsdestotrotz markieren diese Beispiele die ersten Bemühungen bei der Entwicklung von Plattformen für die wissenschaftliche Analyse von Code. Die Lektion ist klar: Jedes Stück Quellcode ist immer nur Teil eines größeren Ganzen.

4. Code ist keine Poesie (meistens jedenfalls)

Loss Pequeño Glazier besteht in seinem Artikel „Code as Language“ darauf, dass, „wenn Sprache als geschriebene Symbole definiert wird, die in Kombinationen und Mustern organisiert sind, um Gedanken und Gefühle auszudrücken und zu kommunizieren – Sprache also ausführt –, auch Code Sprache ist“.⁴⁸ Glazier sieht im Code einen kreativen, sogar literarischen Akt, eine Art des Einschreibens oder des „Denkens durch Gedachtes“. In seinem Essay bietet Glazier zusätzliche Objekte für die CCS-Analyse an, darunter die für eine bestimmte Operation erforderliche Prozessorgeschwindigkeit. Sein Beispiel verweist auf die Doppelnatur des Codes als Aufzeichnung und als Teil eines Echtzeitprozesses. Statt mehr als nur eine Reihe utilitaristischer Befehle zu sein, sendet Code Zeichen von „Humor,

⁴⁸ Loss Pequeño Glazier, „Code as Language“, in: *Leonardo Electronic Almanac* 14:5 (2006).

Innovation, Ironie, Doppelbedeutungen und einer Konzentration auf das Spiel der Sprache. Er bedeutet das Setzen von Zeichen mit einem Sinn für Treffsicherheit.“ Auch hier ist der Schlüssel zu den CCS das Verständnis, dass die Herausforderungen beim Programmieren und Encodieren nicht darin bestehen, den einen richtigen Befehl aus einem Codex von Befehlen zu finden, sondern eine bestimmte Kombination von Zeilen zu wählen (und manchmal zu schaffen), um derart eine Struktur aufzubauen, die ästhetisch, funktional und sogar konzeptionell mit dem anderen Diskurs der codierten Objekte sowie mit dem mathematischen und natürlichsprachigen Diskurs in Resonanz steht und funktioniert.

Wenn Code eine Sprache ist, macht das dann Programme zu Poesie? In „The Aesthetics of Generative Code“ vergleichen Geoff Cox, Alex McLean und Adrian Ward Code mit Poesie und entwickeln einige Techniken zu dessen Interpretation: „Offensichtlich funktioniert Code wie Poesie, indem er mit den Strukturen der Sprache selbst sowie mit unseren entsprechenden Wahrnehmungen spielt.“⁴⁹ Wie Cayley und andere sind sie der Meinung, dass die Bedeutung des Codes mit seiner Ausführung verbunden ist. Ihre Lesart beschränkt sich jedoch nicht auf die Ausführung des Codes, da sie auch „die zweckorientierten Arrangements des Codes durch die Programmierenden“ betonen. Sie stellen zum Beispiel fest, dass sogar die visuelle Darstellung von Code für menschliche Leser:innen gemacht ist, denn „derselbe Code könnte in jeder beliebigen Form oder Anordnung ausgedrückt werden und würde die gleiche Ausgabe liefern“. Nichtsdestotrotz untersuchen sie in einem beispielhaften Schritt von CCS das Sprachspiel eines Perl-Gedichts, in dem der Autor „=“ („ein Vergleichsoperator für Zeichenketten“) anstelle von „eq“ („ein numerischer Ver-

⁴⁹ Geoff Cox, Alex McLean und Adrian Ward, „The Aesthetics of Generative Code“, in: *Generative Art 00*, Milan: Politecnico di Mil 2000. <https://gem.puredata.info/pd/pd/Members/zmoelnig/testing/cox2001aesthetics.pdf> [Letzter Zugriff: 1.9.2023].

gleichsoperator“) verwendet. Durch die genaue Lektüre wird die Bedeutung einer paradigmatischen Wahl für die Gesamtbedeutung des Textes deutlich. Um weiter zu argumentieren, dass Code eine sprachliche Form für menschliche Zuhörer:innen ist, präsentieren sie ihren Aufsatz als „bedingte Aussage“ (*conditional statement*) von Code (wenn, dann), wobei auf ein „wenn“ ihr Abstract als {Bedingung} folgt und der Aufsatzkörper als {Aussage} gekennzeichnet ist. Obwohl sie ein überzeugendes Plädoyer für eine menschliche Poetik in der Konstruktion von Code halten, wäre es übertrieben zu behaupten, dass jeglicher Code einer Ästhetik folgt oder auf Lesbarkeit hinarbeitet. Nichtsdestotrotz kommuniziert der Code stets durch Symbol und Leerraum.

Abgesehen von den wenigen Kunstobjekten, die als Poesie geschrieben wurden, ist der meiste Code Welten von Literatur, Film und bildender Kunst entfernt und ähnelt eher juristischem Code – sein Text ist funktional und in erster Linie darauf ausgelegt, etwas zu tun, statt zu sein, zu performen, statt zu reflektieren. Im Gegensatz zu juristischem Code geht man bei Computercode jedoch in der Regel davon aus, dass er für menschliche Augen unsichtbar ist. Abgesehen von den Fällen, in denen die Maschine Code erzeugt, den seine Programmierer:innen und Benutzer:innen nie zu Gesicht bekommen, wird der Computercode jedoch häufig von anderen Programmierenden gesehen, die ihn überarbeiten müssen. Und selbst wenn der Code nur seinem:r eigenen Coder:in unter die Augen kommt, wie ein weggeschlossenes Tagebuch oder nie abgeschickte Briefe, ist er dennoch ein semiotischer Ausdruck. (Natürlich präsentiert computergenerierter Code Autor:innen als Cyborgs, die freilich mindestens einen Schritt vom produzierten Code entfernt sind. Und fraglos wird der Mensch bei der Interpretation eines solchen Codes Maschinen als Mitwirkende heranziehen müssen.) Noch wichtiger ist – und ich hoffe, dass dies nicht zu vage klingt –, dass die Art und Weise, wie der

Code versucht, eine bestimmte Funktion zu erfüllen, Auswirkungen auf Epistemologien, kulturelle Annahmen über *gender*, *race* und Sexualität, wirtschaftliche Philosophien und politische Paradigmen hat. Diese Aufzählung reicht jedoch nicht ansatzweise aus, um die Angelegenheiten anzuschneiden, die speziell für Computer gelten.

Die Frage der Urheberschaft ist bei der Interpretation von Code viel komplexer als beim traditionellen Schreiben. Code hat häufig mehrere Autor:innen, die meist nicht genannt werden, und große Teile werden aus früherem oder online gefundenem Code übernommen. Außerdem ist ein Teil des Codes kaum mehr als ein Standardtemplate, das manchmal von einer IDE automatisch generiert wird. Die Verwendung weitverbreiteter Algorithmen könnte man mit der Verwendung einer Schraube vergleichen. Mechaniker:innen zitieren nicht jedes Mal den:die Erfinder:in der Schraube, die sie verwenden, obwohl Programmierer:innen gelegentlich Code durchaus bestimmten Quellen zuordnen. Nichtsdestotrotz hat die Literaturanalyse andere Wege gefunden, um mit Fragen der Autorschaft umzugehen, einschließlich der Foucault'schen Vorstellungen, dass Autor:innen selbst eine Ansammlung von Einflüssen sind. Einfach ausgedrückt: Es ist nicht notwendig, die Autor:innen einer bestimmten Zeile zu bestimmen, um das größere Werk des Codes zu interpretieren. Bedeutung braucht keine Fingerabdrücke. Wenn jedoch die Autor:innen ihren Kommentar veröffentlichen, wie in Terry Winograds SHRDLU, um nur ein Beispiel zu nennen, wird dieser Kommentar zu einem Paratext, der die die Bedeutung des Codes formt.

5. Code ist mehr als ein statischer Text

Eines der Probleme bei der Analyse von Code ist, dass sein schriftlicher Ausdruck nur seine statische Form darstellt, bevor

er verarbeitet wurde. Code ist gleichzeitig das, was er ist, und das, was er tut. Das ist nicht unbedingt dasselbe, wie zu sagen, dass Code eine performative Äußerung im Sinne J. L. Austins ist, sondern meint eher, dass der statische Code eine Form seiner Existenz darstellt, bevor er von der Maschine verarbeitet wird, ob er nun kompiliert ist oder nicht. Wie Wendy Chun betont hat, ist der Quellcode nicht dasselbe wie die ausführbare Datei.⁵⁰ Das heißt, das Studium des Codes erfordert nicht nur die Lektüre seiner Symbole, sondern auch ein Verständnis dafür, dass diese Symbole im Laufe der Zeit in dynamischer Wechselwirkung mit anderen Systemen – einschließlich Menschen – Zustandsänderungen in der Software und Hardware verursachen.

Evan Buswell hat Werkzeuge entwickelt, um den Zustand eines Softwaresystems dynamisch zu annotieren, was dem:der Kritiker:in/Leser:in/User:in die Möglichkeit gibt, den Code zum Zeitpunkt der Ausführung – den wir auch als *Software* bezeichnen können – zu kommentieren. Aus der Sicht der Software Studies geben solche Werkzeuge Leser:innen die Möglichkeit, Prozesszustände der Software in Aktion zu identifizieren. Ich betrachte dies jedoch als einen Hauptaspekt beim Lesen des Codes, da dieser Zustand durch die Interaktion der codierten Symbole mit dem System verursacht wurde. Solche Lektüren von Code in Aktion sind aufgrund der Komplexität der Interaktion zwischen dem laufenden Code und dem System (oder anderen Systemen) notwendig – eine Komplexität, die oft zu Zuständen führt, die nicht einfach vorhergesagt werden können.

Betrachten wir im Gegensatz dazu einen Film. Die einzelnen Bilder eines Films (auf Zelluloid) interagieren auf vorhersehbare Weise mit einem bestimmten Projektor. Sicherlich gibt es Schwankungen bei verschiedenen Projektoren (etwa Lampen-

⁵⁰ Chun, *Programmed Visions* (wie Anm. 40).

helligkeit oder eine verringerte Motordrehzahl aufgrund von Verschleiß), aber diese sind vergleichsweise minimal. Die Wirkung der Maschine auf das Medium, das Licht, das durch das Zelluloid scheint, variiert nicht. Außer wenn ein unvorhersehbares Element von außen hinzutritt, erzeugt der Code bei der Verarbeitung auch eine Reihe vorhersehbarer Effekte. Dennoch macht die schiere Komplexität des Systems die Auswirkungen des Codes schwer vorhersehbar – ein Grund, warum die Fehlersuche ein so wichtiger Bestandteil der Programmierung ist. Der Betrieb der Maschine auf dem Medium ist keine geradlinige Aktivität wie die Projektion von Filmen, sondern eine Reihe von interagierenden Prozessen, die durch und auf der Hardware ablaufen. Aus diesem Grund beinhaltet die Analyse von Code eine Interpretation im Saussure'schen Sinne der Zeichen und Signifikanten, aber auch die Analyse dieses sich ständig wandelnden Zustandsverständnisses.

Eine Analogie, die Code und ausgeführter Software vielleicht näherliegt, ist die Betrachtung von Bauplänen. Auch wenn darin alle Elemente aufgeführt sind, gibt ein Bauplan nicht an, wie das fertige Produkt funktionieren wird. Zwar wird jede Plattform den Code auf diskrete und wiederholbare Weise verarbeiten. Die Leistung des so konstruierten Objekts kann jedoch unvorhersehbar sein, insbesondere wenn menschliche Faktoren wie Benutzer:innen oder reale Kräfte wie Hitze oder Wind hinzukommen. Aufgrund dieser Konsistenz, und weil diese Konsistenz die notwendige Bedingung für die Softwareproduktion und -verbreitung ist, hängt die Erstellung und die Lektüre von Code von der Fähigkeit ab, die Zustände der Systeme während der gesamten Dauer der Implementierung (Kompilierung, Verarbeitung und Ausführung) vorherzusagen, zu produzieren und zu testen. Angesichts der Komplexität dieser Systeme erfordert das vollständige Verständnis des Codes ein Verständnis des Codes in Aktion.

6. // Warnende Kommentare

Eine meiner ersten Erfahrungen mit den CCS mag als abschreckendes Beispiel dienen. Als ich einer Gruppe von Programmier:innen eine frühe Version der CCS vorstellte, fragte einer von ihnen, ob ich die CCS demonstrieren könne, indem ich diese Methoden auf ein Programm mit – auch in Bezug auf Eingabe und Ausgabe – geringem natürlichem Sprachinhalt anwende. Der Fragesteller schlug Quicksort vor, einen Algorithmus, der dazu dient, Zahlengruppen schnell zu ordnen, indem man sie in Gruppen aufteilt. In meinen frühen Versuchen, die Noah Wardrip-Fruin mit dem Interpretieren eines Stoppschildes vergleichen würde, benutzte ich Quicksort als Metapher für die soziale Organisation von Gemeinschaften und zog eine Analogie für die Art und Weise, wie Wohnstraßen oder sogar eine Autobahn dazu dienen können, eine Bevölkerungsgruppe aufzuteilen und zu beherrschen; ein Beispiel wäre das rassistische Redlining. Doch obwohl meine Analyse etwas über Nachbarschaftshierarchien aussagte, bot sie kaum Einblicke in Quicksort selbst, noch zog sie aus Quicksort eine Lehre über die Gesellschaft, aus der der Algorithmus stammte. Hier kommt Hofstadters Appell an isomorphe Analogien wieder ins Spiel. Mein Hauptfehler bestand darin, Quicksort losgelöst von seinem historischen, materiellen und sozialen Kontext zu analysieren. Bei einem Algorithmus wie Quicksort trifft der Code am Ort seiner Implementierung und im Kontext der Anwendung, in der er eingesetzt wird, auf die gesellschaftliche Sphäre. Ich habe mich nicht mit den kulturellen Ursprüngen von Quicksort in der Geschichte der Rechenapparate oder gar mit einer bestimmten Inkarnation von Quicksort in einer spezifischen Programmiersprache beschäftigt (ich habe Pseudocode verwendet). Ohne den menschlichen Kontext des Quicksort-Codes in Bezug auf Urheber:innenschaft, Verwendung, Entwicklung, Verbreitung oder Betrieb zu erörtern, blieb mir kaum mehr als ein abstrakter Prozess.

Die Interpretation erfordert die Lektüre eines Objekts in seinem (post-)humanen Kontext durch eine bestimmte kritische Linse. Dieser Kontext umfasst menschliche Maschinen, die in Akteursnetzwerken operieren.⁵¹ So kann einen beispielsweise eine einfache Schleifenfunktion an die Wiederkehr des Verdrängten erinnern, aber wenn diese Metapher keine Bedeutung in Bezug auf den besonderen, materiellen Kontext des Skripts selbst hat, wird die Interpretation eher wie eine Zumutung, wie eine reine Projektion wirken. Das mindert nicht ihren Wert, signalisiert aber eine reduzierte Auseinandersetzung mit der Produktion des Codes und seiner Bedeutung. Findet man jedoch eine rekursive Schleife in einem Programm, das dazu bestimmt ist, seine Benutzer:innen zu psychoanalysieren, mag man eine Verbindung zwischen der Rekursion und der psychoanalytischen Sichtweise der Wiederkehr des Verdrängten herstellen. Obwohl diese Computerprogramme wie jeder kulturelle Text durchaus bedeutungsvoll sind, ergeben sie nur dann einen Sinn, wenn wir ihren materiellen und soziohistorischen Kontext befragen, sowohl den unmittelbaren als auch den umfassenderen, und ihre Zeichen und Systeme vor diesem Hintergrund lesen.

In diesem Plädoyer habe ich durchgehend versucht, die Momente in anderen kritischen Texten hervorzuheben, zu denen spezifische Code-Elemente gelesen wurden, weil dies die Art von Interpretation ist, die in Analysen von Codework und Code am meisten fehlt. Kulturwissenschaftler:innen sprechen oft abstrakt von Programmierpraktiken oder -prozessen, ohne sich mit dem Code selbst zu befassen. Meine Ausführungen sollten jedoch nicht die Notwendigkeit herunterspielen, diese Strukturen mit Blick auf grundlegende menschliche Belange zu interpretieren, die sich auf *race*, *Ethnie*, *gender* und *Sexualität*

⁵¹ Bruno Latour, *Eine neue Soziologie für eine neue Gesellschaft*, Frankfurt a.M.: Suhrkamp 2007 bietet eine umfassende Darstellung der Akteur-Netzwerk-Theorie, die anstelle einzelner menschlichen Subjekte ein Modell von Netzwerken von Maschinen und Menschen vorstellt, die gemeinsam und in Übereinstimmung handeln.

beziehen, auf den militärisch-industriellen, den Unterhaltungs- und den akademischen Komplex, auf Überwachung und Kontrolle über elektronische Systeme, um nur einige zu nennen. Solche Lektüren werden sich als entscheidend für die Analyse von Systemen erweisen, die alles Mögliche leisten sollen – von der Gesichtserkennung über die taktische Simulation politischer Systeme bis hin zur Modellierung des menschlichen Denkens.

Seit ich diese Idee zum ersten Mal öffentlich vorgestellt habe, sträuben sich einige Wissenschaftler:innen gegen die Idee, Code zu lesen. Es gibt einfach zu viel davon, sagen sie. Ich kann nicht auf den Code des Objekts zugreifen, das ich untersuchen will, sagen sie. Auf den ersten Einwand antworte ich wie die kritische Rechtswissenschaft, die nicht versucht, jedes Wort des Gesetzestextes zu analysieren, sondern ihre Energie auf bestimmte Momente und Texte konzentriert. Was den zweiten Einwand betrifft, verweise ich auf die Literaturgeschichte, in der so viele Texte für die Geschichte verloren gegangen sind, doch Gelehrte lesen und interpretieren weiterhin das, was wir besitzen. Nicht jede:r, der:die sich mit digitalen Objekten befasst, wird den Code studieren, aber denjenigen, die es können, sage ich: Sie erwartet viel sinnvolle Kommunikation und engagierte Auseinandersetzung.

7. Ein günstiger Zeitpunkt

Zahlreiche Entwicklungen haben dazu geführt, dass heute der Moment für die Critical Code Studies gekommen ist. Erstens werden höhere Programmiersprachen für eine immer breitere Öffentlichkeit lesbar. Inform 7, eine Sprache für *interactive fiction*, ist nur ein Beispiel von vielen.⁵² Zweitens machen sich

⁵² Daniel Temkin, der den Blog *Esoteric Codes* (<https://esoteric.codes/>) betreibt, empfiehlt die Verwendung von Lobjan, einer Sprache, „die sowohl gesprochen wird, als auch formal ist, mit eindeutigen Bezeichnungen (und Konnotationen!). Es ist die

immer mehr Kulturwissenschaftler:innen (und Künstler:innen) mit der Programmierung vertraut. Drittens beginnt die Coding-Community sich mehr und auf neue Weise für Ästhetik zu interessieren, was zweifellos zu neuen Arten der Programmierkunst führen wird. Viertens: Hochschulen und Universitäten entwickeln immer mehr Programme zur humanistischen Analyse von Wissenschaft und Wissenschaftskultur. Und nicht zuletzt gibt es einfach immer mehr Code in der Kunst und Kunst im Code, *codeworks* und elektronische Literatur; sie alle bieten verbindende Momente zwischen dargestellten und programmierten Zeichen, wie in den Objekten, die Cayley beschrieben hat. Code formt, verändert und begrenzt zunehmend unser Leben, unsere Beziehungen, unsere Kunst, unsere Kultur und unsere zivilgesellschaftlichen Institutionen. Es ist an der Zeit, den Code hinter den Anführungszeichen hervorzuholen, über die Ausführung hinaus zu kommentieren, zu dokumentieren und zu interpretieren. Machen wir den Code zum Text.

Übersetzung Hannes Bajohr

gesprochene Sprache, die einer Programmiersprache am nächsten kommt, und natürlich bestehen fast alle Unterhaltungen daraus, dass sich die Leute gegenseitig korrigieren“ (persönliche Mitteilung, 23. Juli 2019). Er empfiehlt auch, minimalistische Sprachen wie Brainfuck und Ook! zu vergleichen. Beide sind minimalistische Sprachen, aber er merkt an, dass Ook!, das er als „aktiv lesefeindlich“ bezeichnet, visuell schwieriger zu analysieren ist.

KULTURTECHNIK PROGRAMMIEREN. QUELLCODE KRITISIEREN. DREI BEISPIELSZENARIEN

Markus Krajewski

1. Was will und kann Quellcodekritik leisten?

Im Gegensatz zur Forschung und Didaktik der Informatik, bei der es um die Neuentwicklung und effiziente Einübung von algorithmischen Strukturen sowie um die Vermittlung von Prozessen der Software-Herstellung geht, richtet das Forschungsfeld *Programmieren als Kulturtechnik* das Augenmerk auf die Art und Weise, wie Codes entstehen, auf die medialen Praktiken, ihre historischen Umgebungen und wissenstechnischen Bedingungen, die in verschiedenen Epochen der Algorithmenentwicklung vorherrschen. Welche Paradigmen (z.B. objektorientiertes vs. prozedurales Programmieren) und welche informatischen Denkfiguren finden Berücksichtigung? Welche Metaphoriken und Rhetoriken sind dabei involviert? Welche medialen Umgebungen (z.B. im Rahmen einer Geschichte der *Integrated Development Environments*) kommen zum Zuge, und nicht zuletzt, welcher Stilistik folgen und welche Effekte bewirken die einzelnen Befehle im Kontext eines Algorithmus?

Eines der Kernanliegen, Programmieren als Kulturtechnik zu betreiben, besteht jedoch vor allem darin, Algorithmen auch für Nicht-Informatiker:innen verständlich zu machen. Gerade weil Code – auch für Expert:innen – nicht immer leicht zu lesen ist, gerade weil er für Ungeübte mitunter mehr Unverständnis hervorruft, als Einsicht zu bieten, gilt es Wege zu finden, die einer solchen Unverständlichkeit wirksam entgegenarbeiten. Diese Problematik steht am Ausgangspunkt eines Vorschlags, der eine neue Methodik im Umgang mit Code jenseits der

Computerwissenschaften unterbreitet, ein Vorschlag, der sich auch an Code-Affine, aber besonders an philologisch Geübte, an Personen mit ‚klassischer‘ kritischer Lesefähigkeit richtet wie Literaturwissenschaftler:innen, Historiker:innen oder kulturwissenschaftlich Arbeitende. Die Methodik firmiert unter der Bezeichnung ‚Source Code Criticism‘ – oder zu Deutsch: Quellcodekritik – und überträgt klassische Verfahren der Editionsphilologie und Exegese, namentlich das Kommentieren, Kontextualisieren und Erläutern, auf Programmiersprachen und ihre Algorithmen:¹ Den Code eines Softwareprojekts gilt es demnach, noch während seiner Genese – oder auch von späteren Lesern – mit einem ausführlichen, einordnenden und weitere Bezüge herstellenden Kommentar zu versehen, sodass er sich zu einem hybridisierten, selbsterklärenden Artefakt entwickelt, das eine Kombination aus Text und Code gleichermaßen beinhaltet. Dessen Ergebnisse lassen sich idealerweise einerseits in Form eines digitalen ‚Buchs mit Code‘ samt seiner kulturhistorischen und funktionalen Erklärung lesen; andererseits können diese hochverdichteten Text-Code-Hybride als ausführbares Programm, als selbstständige Softwareanwendung zu nutzen sein.

Was unterscheidet Programmieren als Kulturtechnik von der konventionellen Art, Programmieren zu lernen und zu lehren? Kulturtechniken wirken mit ihrem Zusammenspiel von gezielten körperlichen Gesten und dem Einsatz von Objekten wie Werkzeugen, Instrumenten oder anderen Medien auf eine Weise zusammen, die eine spezifische kulturelle Wirkung erzeugt. Das Zusammenspiel von Dachshaarpinsel und einer bestimmten Handhabung dieses Instruments *mit Sprezzatura* bringt eine kulturelle Leistung hervor, die sich etwa in eine europäische Malereitradition einschreibt und sie zugleich

¹ Markus Krajewski, „Against the Power of Algorithms. Closing, Literate Programming, and Source Code Critique“, in: *Law Text Culture* 23 (2019), S. 119–133.

modifiziert und voranzubringen vermag. Die Kulturtechnikforschung untersucht daher die Praktiken jener Prozesse, die in Bezug auf ihre Medialität kulturkonstitutiv sind, einschließlich der an ihrer historischen Entwicklung beteiligten Prozeduren, Gesten und Werkzeuge sowie ihrer kulturellen und epistemischen Grundlagen.² Kulturtechniken verfügen dabei immer auch über eine ästhetische Komponente, die über die reine Funktionalität der Praktiken hinausgeht. Dies gilt auch im Kontext der Computer Science, wenn beispielsweise einer ihrer Gründerväter, der Informatiker Donald Knuth, vom Prozess des Programmierens fordert, bei der Entwicklung von Code ebenso Fragen des Stils, der Eleganz und nicht zuletzt der Schönheit zu berücksichtigen.³

Trotz des gestiegenen Interesses an der Erforschung von Kulturtechniken und ihrer wachsenden akademischen Bedeutung sind Fragen nach den digitalen Praktiken wie der Entwicklung von Algorithmen und ihrer Funktionsweise als Kulturtechnik bisher weitgehend außer Acht gelassen worden.⁴ Diesem Desiderat will das Verfahren der Quellcodekritik ebenso begegnen, wenn es Codieren als Kulturtechnik beschreibt. Denn dazu gehört nicht nur die Fähigkeit, Code zu lesen und zu schreiben oder ihn für den alltäglichen Softwarebedarf neu zu entwi-

² Erhard Schüttpeitz, „Die medienanthropologische Kehre der Kulturtechniken“, in: *Archiv für Mediengeschichte* 6 (2006), S. 87–110; Bernhard Siegert, „Cultural Techniques: Or the End of the Intellectual Postwar Era in German Media Theory“, in: *Theory, Culture & Society* 30:6 (2013), S. 48–65; Bernhard Siegert, *Cultural Techniques. Grids, Filters, Doors, and other Articulations of the Real*, New York: Fordham University Press 2015.

³ Donald E. Knuth, „Computer Programming as an Art“, in: *Communications of the ACM* 17:12 (1974), S. 667–673.

⁴ Zu einem thematischen Überblick über das Forschungsfeld siehe Geoffrey Winthrop-Young, „The Kultur of Cultural Techniques. Conceptual Inertia and the Parasitic Materialities of Ontologization“, in: *Cultural Politics* 10:3 (2014), S. 376–388; Geoffrey Winthrop-Young, „Discourse, Media, Cultural Techniques: The Complexity of Kittler“, in: *Modern Language Notes* 130:3 (2015), S. 447–465; Geoffrey Winthrop-Young, „Siren Recursions“, in: Stephen Sale und Laura Salisbury (Hg.), *Kittler Now: Current Perspectives in Kittler Studies*, Cambridge, MA: Polity Press 2015, S. 71–94; Geoffrey Winthrop-Young, „The Kittler Effect“, in: *New German Critique* 44:132–3 (2017), S. 205–224.

ckeln. Darüber hinaus zählt dazu, den Code in seinen diskursiven und historischen Zusammenhängen einer kritischen Analyse zu unterziehen, um ihn durch diese Einordnungen und Kontextualisierungen besser verständlich zu machen. Das Schlüsselement in diesem Verfahren ist der Kommentar.

Quellcodes verfügen prinzipiell über zwei verschiedene Ebenen. Auf der ersten Ebene liefern sie eine Reihe von Befehlen in einer Programmiersprache, die von der Maschine sequentiell abgearbeitet werden. Auf der zweiten Ebene lassen sich zwischen diese Befehle wiederum Texte einfügen, die dank vorangestellter Markierungen durch bestimmte Zeichen nicht als Befehle ausgeführt, sondern als Kommentare verstanden werden, die nicht in den Binärcode, also in das von der Maschine ausführbare Programm, übernommen werden. Dank dieser Staffelung in zwei Ebenen lassen sich Codes durch solche Kommentare gleichsam domestizieren oder (ein-)hegen, das heißt, sie durch eine unmittelbare Diskursivierung der einzelnen Anweisungen im Quellcode, durch Einordnungen ihrer Funktionsweise zu erläutern und verständlich zu machen.⁵ Konkret bedeutet dies für kulturwissenschaftliche Analysen, dass sie durch die Lektüre der Kommentare lernen können, eine kritische Distanz zu Algorithmen einzunehmen. Und schließlich umfasst dies ebenso, mithilfe der Erläuterungen nicht nur die Funktionsweise der Codes zu decodieren und ihre Wirkung, sondern ebenso ihre sprachlichen Besonderheiten, ihr Design und ihren Stil zu verstehen und weitergehend zu untersuchen, sodass ein tiefergehendes Verständnis gelingt, um sie nach politischen, datenschutzrechtlichen, aber auch moralischen oder kulturellen Standards taxieren zu können. Dieser Ansatz unterscheidet sich in seiner Stoßrichtung von vergleichbaren Konzepten im Bereich der Software-Studien, wie sie etwa

⁵ Markus Krajewski, „Hilfe für die digitale Hilfswissenschaft. Von den Digital Humanities verspricht man sich wahre Wunder, obwohl sie nur eine einfache Hilfswissenschaft sind“, in: *Frankfurter Allgemeine Zeitung* 85:4 (2019).

von Matthew Kirschenbaum oder von Stephen Ramsay, Lev Manovich, Matthew Fuller oder Nick Montfort entwickelt worden sind, auch wenn sich alle Ansätze in dem übergeordneten Ziel treffen, die digitale Kompetenz von Geisteswissenschaftler:innen zu stärken.⁶ Insbesondere Mark C. Marino hat in der Entwicklung des innovativen Feldes der *Critical Code Studies* gezeigt,⁷ wie eine hermeneutische Lektüre von Algorithmen funktionieren kann, zum Beispiel in seinen *close readings* der kollaborativen Geschichte *exquisite_code*,⁸ in jüngerer Zeit durch seine Erläuterungen zu Friedrich Kittlers Codeproduktion und nicht zuletzt in seinem programmatischen Critical Code Studies-Manifest (vgl. Marinos Beitrag im vorliegenden Band).⁹ Darin fordert er, vom Code nicht nur „metaphorisch“ zu sprechen, sondern ihn „wie Text zu analysieren und zu erklären – als Zeichensystem mit eigener Rhetorik, als semiotische Kommunikation.“¹⁰

Die Kulturtechnik Programmieren setzt komplementär dazu auf einer pragmatischen Ebene an, wenn es darum geht, die Folge der Befehle in einem Programm durch extensive Kom-

⁶ Matthew Kirschenbaum, „Hello Worlds. Why humanities students should learn to program“, in: *Chronicle of Higher Education* 55:20 (2009); Matthew Kirschenbaum, „What Is Digital Humanities and What’s It Doing in English Departments?“, in: *ADE Bulletin* 47:150 (2010), S. 55–61; Stephen Ramsay, *Reading machines. Toward an algorithmic criticism*, Urbana: University of Illinois Press 2011; Lev Manovich, *Software takes command*, New York: Bloomsbury 2013. Siehe auch Geoff Cox und Christopher Alex McLean, *Speaking code. Coding as aesthetic and political expression*, Cambridge, MA: The MIT Press 2013; David M. Berry und Anders Fagerjord, *Digital Humanities. Knowledge and Critique in a Digital Age*, Oxford: Polity Press 2017; Daniel Punday, *Computing as Writing*, Minneapolis: University of Minnesota Press 2015, Kap. 3.

⁷ Mark C. Marino, „Critical Code Studies“, in: *Electronic Book Review*, Dezember 2009, <http://electronicbookreview.com/essay/critical-code-studies/> [Letzter Zugriff: 17.7.2023]; Mark C. Marino, „Critical Code Studies and the electronic book review: An Introduction“, in: *Electronic Book Review*, September 2010, <http://electronicbookreview.com/essay/critical-code-studies-and-the-electronic-book-review-an-introduction/> [Letzter Zugriff: 17.7.2023].

⁸ Mark C. Marino, „Reading ‚exquisite_code‘“, in: Katherine N. Hayles und Jessica Pressman (Hg.), *Comparative textual media. Transforming the humanities in the postprint era*, Minneapolis: University of Minnesota Press 2013, S. 283–309.

⁹ Mark C. Marino, *Critical code studies*, Cambridge, MA: The MIT Press 2020, S. 161–197.

¹⁰ Im vorliegenden Band auf S. 31.

mentare verständlich zu machen. Zugleich setzt die Kulturtechnik Programmieren jedoch auch auf einer historischen Tiefenebene an, wenn es darum geht, die Codes in ihrer Genese, als Denkfiguren und Metaphern mit ihrer jeweils eigenen Historizität zu entfalten. Kurzum, die Kulturtechnik Programmieren trägt Verfahren der Medienarchäologie auf beziehungsweise zwischen die Ebenen der Codes, um deren Befehle und Funktionen auf diese Weise diskursanalytisch zu fassen.

2. Drei Beispiele

Wie genau kann Quellcodekritik auf einer praktischen Ebene aussehen? Im Folgenden sei das Verfahren anhand von drei Code-Fragmenten aus unterschiedlichen Kontexten skizziert (siehe Listing 1 auf S. 71, Listing 2 auf S. 75 und Listing 3 auf S. 84), um sowohl die Varianz als auch das Potential dieses Ansatzes zu konturieren.

2.1 Fibonacci verstehen

Das erste Beispiel in Listing 1 befasst sich mit einer mathematischen Denkfigur, deren Verwendung in der abendländischen Geistesgeschichte recht genau datiert werden kann, ist sie doch so alt wie die Ankunft des indo-arabischen Zahlensystems in Europa: die Rekursion. Eines der bekanntesten Beispiele für eine Rekursion ist eine Folge von natürlichen Zahlen, die sogenannte Fibonacci-Reihe, deren jeweils nächstes Glied n aus der Summe des Vorgängers $n - 1$ und des Vorvorgängers $n - 2$ gebildet wird. Um also das Ergebnis für ein beliebiges n zu berechnen, zum Beispiel $n = 7$, muss zunächst das Ergebnis von $n = 6$ berechnet werden, wofür zunächst das Ergebnis von $n = 5$ berechnet werden muss, wofür zunächst das Ergebnis von $n = 4$

berechnet werden muss ... usw. bis zu $n = 1$. Dieser Algorithmus wird in dem von Leonardo Pisano, genannt Fibonacci, 1202 publizierten Rechenlehrbuch namens *Liber Abaci* vorgestellt.¹¹ In Listing 1 wird in den Zeilen 224–240 kurz die kulturhistorische Rahmung der Fibonacci-Zahlen einschließlich eines Originalzitats von Fibonaccis Algorithmus aus dem *Liber Abaci* vorgestellt (die berühmten Kaninchen), bevor dieser Algorithmus dann tatsächlich in Zeile 246 zur Ausführung gelangt. Zunächst jedoch wird in Zeile 244 die Anzahl der Durchläufe (oder Generationen von Kaninchen) mit n explizit festgelegt, denn jede Rekursion benötigt ein klar definiertes Ende.

Der tatsächliche Aufruf erfolgt also in Zeile 244, begleitet von einem berühmten Zitat zur Rekursion von L. Peter Deutsch, das für den ausführbaren Code zwar unwirksam bleibt, auf der erläuternden Ebene jedoch unmittelbar die besondere Eleganz der algorithmischen Struktur hervorhebt.

Nach diesem ‚Weckruf‘ springt das Programm dann von Zeile 244 direkt in die entsprechende Methode in Zeile 248, um die Fibonacci-Zahl für die 11. Generation zu berechnen. Zunächst wird überprüft, ob die Rekursion schon zum Ursprung, also bis zur 1. Generation zurückgerechnet hat. Da 11 jedoch größer als 1 ist, bleibt die Bedingung in Zeile 253 falsch und das Programm springt in Zeile 258, wo sich dieselbe Methode nun gleich zweimal erneut selbst aufruft (also direkt wieder zurück in Zeile 248 springt, alle Befehle abarbeitet bis Zeile 258, dann wieder zurück zu Zeile 248, so lange, bis die Bedingung in Zeile 253 wahr wird, also bis zu $n = 1$ zurückgerechnet wurde). Diese Schleife wird einmal durchlaufen für die vorherige und sodann erneut ein

¹¹ Leonardus Pisano, *Fibonacci's Liber abaci. A translation into modern English of Leonardo Pisano's Book of calculation*, New York: Springer 1202/2002, S. 404. Für Weitergehendes zur Rekursion in ihrer kulturhistorischen und wissenschaftlichen Entwicklung siehe Markus Krajewski, „Die Rekursion. Zur Wissensgeschichte einer Denkfigur“, in: Ines Barner und ders. (Hg.), *Rekursion und Revision. Robert Walser, Christian Morgenstern und die mimetische Praxis des Schreibens*, Paderborn, Leiden: Brill-Fink 2022, S. 17–72.

zweites Mal für die vorvorherige Generation. Die Befehle werden innerhalb der Schleife nunmehr jedes Mal gesondert und Schritt für Schritt abgearbeitet, so lange, bis mit $n = 1$ jeweils die erste Generation erreicht ist. Erst dann sind alle zuvor noch unbekanntenen Werte berechnet, das Programm kann nun alles aufaddieren und zurück in Zeile 244 springen, um den entsprechenden Wert, nämlich 144, auszugeben. Auf dem Bildschirm erscheint sodann folgende Nachricht:

```
In generation 11 there are 144 rabbits.
```

Eine lauffähige Version dieses *JAVA*-Codes, der mit der Integrated Development Environment (IDE) *Apache NetBeans* erstellt wurde, findet sich unter github.com/nachsommer/sourcecode-criticism.

2.2 Jupyter-Monde und digitale Geschichte

Das zweite Beispiel wechselt sowohl die Programmiersprache als auch die digitale Umgebung. Statt *JAVA* und *Apache Netbeans* kommen nun *Python* und *Jupyter Notebook* zum Einsatz. Letzteres ist weniger eine universelle Entwicklungsumgebung als vielmehr eine auf dem Client-Server-Prinzip basierte Web-Umgebung speziell zur Auswertung wissenschaftlicher Daten, quantitativer Analysen sowie zur Implementierung statistischer Methoden. ‚Jupyter‘ ist dabei einerseits als ein Akronym der hauptsächlich verwendeten Programmiersprachen *Julia*, *Python* und *R* zu verstehen. Andererseits spielt es auf Galileis *Sidereus Nuncius* von 1610 an, einer zum Buch gewordenen Notizsammlung von Planetenbeobachtungen, die Galilei mittels der neusten Hochtechnologie seiner Zeit, das heißt mit selbst entworfenen und optimierten Teleskopen, hat anfertigen können. Neben der medientheoretisch aufschlussreichen

```

223 private void calculateSequence() {
224     /* What seems so interesting about calculating this series of numbers? There are two
225        reasons: one is the history of this algorithm, the other is the style — how this
226        algorithm can be implemented.
227
228        1. In 1202, Fibonacci published the Liber Abaci, introducing not only the arabic
229        numerals, but also the concept of the algorithm to Western mathematics. It
230        contains a well-known series of numbers named after him. He developed his
231        example in a famous scenario:
232
233        "How Many Pairs of Rabbits Are Created by One Pair in One Year.
234        A certain man had one pair of rabbits together in a certain enclosed place, and
235        one wishes to know [...]" \parencite[404]{fibonacci:1202}
236
237        2. The sequence of numbers can be calculated in two different ways. In the first,
238        this is done most elegantly by a recursive method, starting with the last
239        generation and going back to the first, while it calls itself as a method while
240        it calls itself as a method while it calls itself as ... until the initial value
241        n=1 is reached. Then all the numbers are available and can be added to the final
242        sum.
243
244        The second way is to calculate the value simply by remembering the last and the second
245        to last value.
246
247        This alternative is rather easy in terms of calculating power and resources, however it
248        requires more code to produce the non-recursive algorithm.
249
250        In the following, both ways are implemented, so the user can compare the differences by
251        executing the code line by line with a debugger...
252
253        */
254
255        // Begin calculating ... for n=: you name it ...
256        int n = 11;
257
258        // "To iterate is human, to recurse divine" (L. Peter Deutsch)
259        System.out.println("In generation " + n + " there are " + fibo(n) + " rabbits.\n");
260    }
261
262    private long fibo(int n) {
263        /* This code fragment is rather short, because it makes use of a specific trick. The
264           method fibo(n) calls itself within its definition: a classical recursion. This
265           is most elegant, because the calculation can be written in only one line.
266
267           However, this elegance comes with a huge computational cost.
268
269           */
270        if (n <= 1) {
271            return n;
272        } else {
273            /* This is the crucial line: The return value of the function is a double call of the
274               function itself. Before returning the value for n it has to calculate the sum of
275               n1 and n2, where n denotes the generation of rabbits.
276
277               */
278            return fibo(n - 1) + fibo(n - 2);
279        }
280    }

```

Listing 1:
Eine Rekursion, die Fibonacci-Zahlen
und ihre Erklärungen liefert

Transformation des Teleskops von einem Instrument in ein Medium, die Joseph Vogl herausgearbeitet hat,¹² zeichnet sich der Text durch einen ständigen Wechsel der Ebenen zwischen Beschreibung und Reflexion, zwischen Narration und Interpretation, zwischen Daten und Metadaten, zwischen Code und Kommentar aus, etwa wenn Galilei zunächst beschreibt, was er beim Blick durch das Teleskop zum Mond sehen kann, was zuvor nicht sichtbar war, bevor er dann auf einer übergeordneten Ebene mit den Gesetzen der Optik und Geometrie kommentiert, wie diese neue Sichtbarkeit mithilfe des Mediums Teleskop erzeugt worden ist. Oder wenn Galilei zunächst über viele Seiten die Konstellation der Jupitermonde relativ zu den Beobachtungszeiten in Zahlen, Worten und Diagrammen festhält bzw. codiert, bevor er daraus auf übergeordneter Ebene seine Meta-Analyse der Jupitermond-Bewegungen kommentierend ableitet.¹³

Dieser gedanklichen Oszillationsbewegung wird seitens der *Jupyter Notebooks* in deren spezifischen Format (*.ipynb) Rechnung getragen: Jeder Absatz wird als eigene Einheit (*cell*) begriffen, die einerseits konventionellen Text im Markdown-Format enthält. Andererseits lassen sich darin ebenso mühelos Code-Fragmente einbetten, die dazu dienen, Bilder einzubinden, Diagramme zu erzeugen, externe Messdaten heranzuziehen, um sie tabellarisch darzustellen. Und schließlich lassen sich aus dieser Umgebung heraus auch externe Prozesse starten und kontrollieren, etwa zum Training neuronaler Netze oder anderer statistischer Analyseprozesse.

Den beständigen Wechsel der Ebenen zwischen Code und Kommentar, zwischen Erzählung und Interpretation erhebt bei-

¹² Vgl. Joseph Vogl, „Luxus“, in: Karlheinz Barck u.a. (Hg.), *Ästhetische Grundbegriffe. Historisches Wörterbuch in sieben Bänden*, Band 3, Stuttgart, Weimar: J.B. Metzler 2001, S. 694–716.

¹³ Vgl. Galileo Galilei, *Sidereus Nuncius. Nachricht von neuen Sternen; Dialog über die Weltsysteme (Auswahl); Vermessung der Hölle Dantes; Marginalien zu Tasso*, hrsg. von Hans Blumenberg, Frankfurt a.M.: Suhrkamp, (1610/2002).

spielsweise das *Journal of Digital History*, das seit 2021 an der Universität Luxemburg erscheint, zum Produktionsprinzip. Einerseits greift die Zeitschrift auf *Jupyter Notebook* als Schreibumgebung zurück. Das heißt, die zur Veröffentlichung eingereichten Aufsätze müssen originär im Format der Jupyter-Notebooks verfasst sein, sodass bereits bei der Produktion des Beitrags zwingend auch auf der Ebene des Quellcodes gearbeitet wird. Das zeigt sich dann etwa in der Darstellung der Beiträge auf der Journal-Webseite, wo nicht nur eine Zeilenzählung zur besseren Referenzierbarkeit mitgeführt wird, sondern eben auch jede visuelle, auditive, multimediale, digitale Quelle mit ihrem entsprechenden *source code* implementiert ist (vgl. Abb. 1 sowie den dazugehörigen Quellcode in Listing 2). Einerseits lässt sich also einfach nur das Bild des Kinora in *figure 9* betrachten.¹⁴ Andererseits kann die Leser:innenschaft sich die Einbindung des Bildes vor Augen führen, mithin sogar die Generierung des Bildes selbst, sofern es nicht einfach eine Pixelgrafik-Datei, sondern eine generative Darstellung in Form einer Vektorgrafik mit ihrerseits programmiertem Quellcode umfasst.¹⁵ Zu diesem Zweck ist unterhalb der Abbildung eine Schaltfläche angebracht, mit deren Aktivierung ein schwarzer Kasten aufscheint, der den Code enthüllt. Die vermeintliche Blackbox der Codierung lässt sich auf diese Weise also sehr einfach öffnen.

Darüber hinaus unterscheidet jeder Aufsatz im *Journal of Digital History* seinerseits noch zwischen zwei Ebenen, und zwar dem *narrative* und dem *hermeneutics layer*. Die Dichotomie von Text und Kommentar oder Daten und Meta-Daten wird damit um eine weitere Iterationsebene verschachtelt: Die Web-Darstellung der einzelnen Beiträge erlaubt es, zwischen beiden Ebenen zu

¹⁴ Tim van der Heijden und Claude Wolf, „Replicating the Kinora: 3D modelling and printing as heuristics in digital media history“, in: *Journal of Digital History* 2:1 (2022), Punkt 82.

¹⁵ Vgl. Benedikt Groß, Hartmut Bohnacker, Julia Laub und Claudius Lazzaroni, *Generative Design Revised. Visualize, Program, and Create with JavaScript in P5.js*, New York: Princeton Architectural Press 2018.

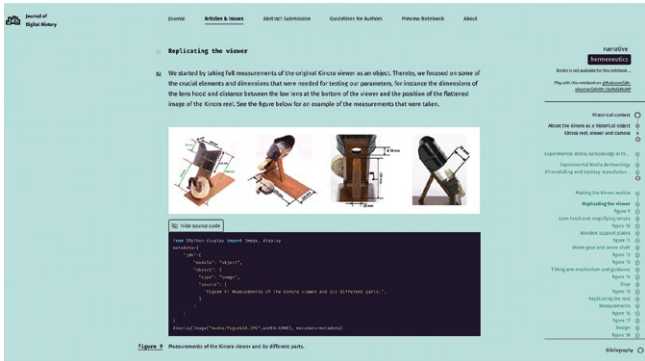


Abb. 1:
Code und Kommentar auf verschiedenen Ebenen, ein Beitrag aus dem *Journal of Digital History*

wechseln, man kann also die Analyse einerseits auf der Ebene der Diegese – hier *narrative* genannt – lesen. Andererseits kann man den Beitrag auf der Ebene des *making-of*, der weiterführenden Anmerkungen oder Selbstinterpretation – hier *hermeneutics* genannt – nachverfolgen.¹⁶ Je nach Beitrag fallen die Einordnungen auf der Meta-Ebene des Kommentars naturgemäß eher zurückhaltend aus, es überwiegt also notwendigerweise die Ebene der Erzählung. So wie im Code für gewöhnlich der Anteil der Befehle den Anteil der Kommentare übersteigt.

¹⁶ Frédéric Clavert und Andreas Fickers, „On pyramids, prisms, and scalable reading“, in: *Journal of Digital History* 1:1 (2021). Die Konzeption nimmt einen Vorschlag von Robert Darnton von 1999 auf und transponiert ihn auf die Ebene einer rein digitalen Zeitschrift. Für eine Kurzdarstellung der Ebenen siehe ebd., Abbildung 2.

```

223 {
224   "cell_type": "markdown",
225   "metadata": {
226     "tags": [
227       "hermeneutics"
228     ]
229   },
230   "source": [
231     "### Replicating the viewer"
232   ]
233 },
234 {
235   "cell_type": "markdown",
236   "metadata": {
237     "tags": [
238       "hermeneutics"
239     ]
240   },
241   "source": [
242     "We started by taking full measurements of the original Kinora viewer as an object.
      Thereby, we focused on some of the crucial elements and dimensions that were needed
      for testing our parameters, for instance the dimensions of the lens hood and distance
      between the low lens at the bottom of the viewer and the position of the flattened
      image of the Kinora reel. See the figure below for an example of the measurements
      that were taken."
243   ]
244 },
245 {
246   "cell_type": "code",
247   "execution_count": 10,
248   "metadata": {
249     "tags": [
250       "hermeneutics",
251       "figure-9"
252     ]
253   },
254   ...
255   "source": [
256     "from IPython.display import Image, display\n",
257     "metadata={\n",
258     "  \"jdh\":\n",
259     "    \"module\": \"object\",\n",
260     "    \"object\": \n",
261     "    \"type\": \"image\",\n",
262     "    \"source\": \n",
263     "    \"figure 9: Measurements of the Kinora viewer and its different parts.\",\n",
264     "    }\n",
265     "  }\n",
266     "]\n",
267     "display(Image(\"media/figure10.JPG\", width=1000), metadata=metadata)"
268   ]
269 },
270 ...
271

```

Listing 2:
Publizieren im *Journal of Digital History* heißt,
im Quellcode schreiben.

2.3 Warburgs Nachbarschaften

Das dritte Beispiel in Listing 3 greift ein altes informatisches Problem auf, nämlich die Erzeugung von Zufallszahlen, was weniger trivial ist, als es auf den ersten Blick erscheinen mag. Zumindest nicht für eine deterministische Rechenmaschine, die keinen echten Zufall aus sich heraus produzieren kann.

Entsprechend startet dieser Code mit einer wissenschaftlichen Forschungsfrage, wie sie neuerdings in Dialogsituationen mit generativer KI wie etwa ChatGPT virulent geworden ist: Wie kann es der Maschine eigentlich gelingen, bei Benutzer:innen Überraschung zu erzeugen? (Dahinter verbirgt sich nicht zuletzt die Frage nach dem Verhältnis von maschineller Kreativität und Zufall.) Was im Fall von Textgeneratoren ganze Forschungsfelder absteckt, wird hier zunächst als Problem eingeeht auf die ungleich überschaubarere Frage, wie eine Rechenmaschine Zufallszahlen erzeugt, um mit dieser Zahl dann aus einer Menge, etwa an Karteikarten, eine akzidentielle Auswahl zu treffen (Zeile 450). Allerdings zeigt sich, dass hier kein echter Zufall entsteht, sondern lediglich simuliert wird. Man spricht daher von Pseudo-Zufall.

Dieser Art des maschinellen Pseudo-Zufalls wird sodann eine alternative Methode gegenübergestellt, und zwar der ‚gelenkte Zufall‘, besser bekannt als *serendipity*. Als historisches Beispiel, anhand dessen diese Methode algorithmisch implementiert wird, dient Aby Warburgs berühmtes „Gesetz der guten Nachbarschaft“, das er als Grundlage seiner auf diese Weise dynamisierten Bibliotheksordnung entwickelt hat.¹⁷ Nach einer konzeptionellen Unterscheidung, wie sich Bücher infolge ihres Inhalts immer wieder neu produktiv zueinander gruppieren lassen, werden drei verschiedene Ansätze thema-

¹⁷ Mario Wimmer, „The Afterlives of Scholarship: Warburg und Cassirer“, in: *History of Humanities* 2:1 (2017), S. 245–270.

tisiert (Zeile 458, 459 und 460), wie sich inhaltliche Ähnlichkeiten zwischen Büchern berechnen lassen, bevor diese dann ab Zeile 463 implementiert werden.

Während das Beispiel aus Listing 1 noch vergleichsweise wenig erläuternde Kommentare umfasst, die Anmerkungen demnach eher als *Parerga* denn als *Ergon* – Letzteres bleibt die Ebene der Befehle – einzuordnen sind, und derweil die Kommentare in Listing 2 sich eher funktional als vertiefend lesen lassen, weist das Beispiel in Listing 3 den Weg, wie im Quellcode die auszuführenden Befehle beinahe zum Beiwerk und die Kommentare dagegen *zum eigentlichen Text* geraten. Dieser enthält nicht nur größere narrative Anteile und unterstützt die üblichen typografischen Auszeichnungen wie Fettdruck bei Zwischenüberschriften oder Kursivierungen bei hervorzuhebenden Passagen. Mehr noch, der Kommentar verfügt über alle notwendigen Elemente, die ein philologischer Apparat erfordert: Neben einer Zeilenzählung wird die Bibliografie selbstständig mitgeführt (vgl. Zeile 240 oder 257, wo die einzelnen Referenzen in diesem Fall via *biblatex* automatisch generiert und verlinkt werden), und im Bedarfsfall lassen sich auf der Ebene der Anmerkungen noch zusätzliche Kommentare in Form von Fußnoten einbetten (vgl. Zeile 460).

Damit hebt sich das Verfahren einer Quellcodekritik nicht nur von ähnlichen Ansätzen innerhalb der *Software* und *Critical Code Studies* ab, indem es komplementär zu deren entkoppelter Hermeneutik die diskursanalytische Beschreibung direkt in den Code einbettet und methodisch Code und Kommentar zu einer neuen, wechselseitigen Bezüglichkeit bringt. Darüber hinaus unterscheidet sich der Ansatz der Quellcodekritik ganz grundsätzlich von den in den Digital Humanities weit verbreiteten Methoden quantitativer statistischer oder

numerischer Analysen, die z.B. nach der Worthäufigkeit fahnden, Motivsuche mit *topic modeling* oder automatisiertes *tagging* betreiben oder aber Stilanalysen anhand von stochastischen Mitteln unternehmen. Im Gegensatz zu diesen Ansätzen zielt Programmieren als Kulturtechnik vielmehr darauf ab, eine qualitative Kernkompetenz der Geisteswissenschaften, das kritische Lesen, in die digitale Welt zu übertragen. Ein solches Verfahren ermöglicht es, innerhalb einer einzigen Datei des Quellcodes, Algorithmen nicht nur verstehen, einordnen und verändern zu können, sondern sie auch mit den Termini, Theorien, Konzepten und Dispositiven der Literaturkritik zu erfassen und zu kommentieren. Diese Form der Kontextualisierung macht es möglich, den Code wie einen literarischen Text zu dekonstruieren oder aber wie eine historische Quelle diskursanalytisch zu fassen.

In dieser Methode fusionieren demnach zu einem bestimmten Grad klassische Literaturanalyse und historische Quellenkritik,¹⁸ die neben einer elaborierten Form des Lesens ebenso eine sorgfältige Untersuchung und Aufbereitung des Materials, in diesem Fall statt mittelalterlicher Urkunden also Algorithmen, mit einer theoretisch fundierten Einordnung verbindet. Konzeptionell umfasst dieser Ansatz daher zum einen die Bereitstellung von Quellcode in Open-Access-Repositories wie *github* oder aber, etwas komplizierter, legt er durch Verfahren wie Dekompilierung oder *reverse engineering* einen Blick in die Ausgangslage der Programme frei. Darüber hinaus beinhaltet die Quellcodekritik die kritische Lektüre von Code, der in seinen dynamischen Transformationen *wie historische Quellen* behandelt werden kann, die in diesem Fall jedoch nicht etwa statisch sind, sondern nicht zuletzt aufgrund ihrer vielen Ver-

¹⁸ Daniela Saxer, *Die Schärfung des Quellenblicks. Forschungspraktiken in der Geschichtswissenschaft 1840–1914*, Oldenbourg, München: De Gruyter, S. 376ff.

sionen eine weitergehende Einordnung und Kommentierung erfordern.¹⁹

Im Gegensatz zu diesem neuartigen Ansatz ist die gegenteilige Idee – Software zu entwerfen, als wäre sie Literatur – keineswegs neu. Der oben bereits erwähnten Donald E. Knuth, Autor des epochalen Buchs *The Art of Computer Programming* (1968–2025) und des Satzsystems T_EX, hatte bereits 1984 einen Artikel mit dem bewusst zweideutigen Titel *Literate Programming* veröffentlicht, in dem er vorschlug, Quellcode von vornherein so zu schreiben, dass er mehr als nur die Befehle der jeweiligen Programmiersprache enthält.²⁰ Als Alternative forderte Knuth eine Metaebene, in der die Entwickler:innen die einzelnen Befehle ausführlich beschreiben, ihren Code also gleich einem Essay verfassen. Der Quellcode würde dann nicht nur die einzelnen Befehle und Datenstrukturen enthalten, sondern diese auch dokumentieren, idealerweise in einer literarisch elaborierten Form. Auf diese Weise würden Algorithmen transparent gemacht werden können – nicht nur für ihre Autor:innen, sondern auch für spätere Leser:innen und Entwickler:innen. Es ist kaum nötig zu erwähnen, dass dieses Paradigma in der täglichen Routine professioneller Softwareentwickler:innen kaum Anwendung findet.

Literatur, Code und ihre Kritik liegen strukturell näher beieinander, als es auf den ersten Blick erscheinen mag. So wie die Kulturtechnik des Lesens auf akademischem Niveau es Philolog:innen ermöglicht, die Art und Weise, wie ein Text konstruiert wurde, offenzulegen, seine Rhetorik, seine Affekterzeugung, seinen Stil und seine literarischen Denkfiguren zu reflek-

¹⁹ Zur Geschichte der Versionierungen und ihrer medialen Praktiken siehe Markus Krajewski, „branch, diff, merge. Versionskontrolle und Quellcodekritik“, in: Jörg Paulus, Andrea Hübener und Fabian Winter (Hg.), *Duplikat, Abschrift & Kopie. Kulturtechniken der Vervielfältigung*, Wien, Köln, Weimar: Böhlau 2020, S. 22–40; Markus Krajewski, „Versionskontrolle“, 2020, <https://github.com/nachsommer/Versionskontrolle> [Letzter Zugriff: 17.7.2023].

²⁰ Donald E. Knuth, „Literate Programming“, in: *The Computer Journal* 27 (1984), S. 97–111.

tieren, so kann die Kulturtechnik Programmieren ihre Praktiker:innen befähigen, Algorithmen weitergehend zu verstehen und historisch zu verorten. Mithilfe der Erläuterungen und Kommentare werden sie in die Lage versetzt, die Funktionsweise der Algorithmen nachzuvollziehen, ihre Entwicklung einzuordnen, ihrerseits auf problematische Stellen hinzuweisen oder Unverständliches gleich im Quellcode zu markieren oder zu emendieren. Ein solcher Fokus schließt die Frage nach der Poetik ebenso wie nach der Poetologie von Algorithmen ein. Bekanntlich gibt es viele Wege, um ein bestimmtes digitales Problem zu lösen. Einige davon sind so überraschungsarm wie die Fahrt über eine zwölfspurige Autobahn durch die Bay Area, andere haben das Entdeckungspotenzial des Feldwegs in Arno Schmidts *Zettel's Traum*, und einige seltene Wege sind sogar wie der noch in die Wildnis zu schlagende Pfad in Knut Hamsuns *Segen der Erde*, mit dessen Bahnung sich das Gestrüpp in zivilisierte Zukunft verwandelt. Kurzum, bei der Klassifizierung und hermeneutischen Lektüre von Codes ist es von entscheidender Bedeutung, auch die Absichten, Formulierungen und Stilfragen zu untersuchen, die bei der Produktion der Codes eingeflossen sind. Denn um zu erkennen, wie Benutzer:innen von Software benutzt werden, ist es notwendig, die Konstruktion und das Design von Algorithmen zu entschlüsseln und sie einer kritischen Reflexion zu unterziehen. Dieser Prozess der kritischen Lektüre und erläuternden Kommentierung von Algorithmen sowie das Engagement für eine geisteswissenschaftliche Codierkompetenz auf der grundlegendsten Ebene des Codes ist, kurz gesagt, das, was *Quellcodekritik* ausmacht.

3. Die Macht der Kommentare

Insbesondere der Kommentar stellt den zentralen Einsatz dar, ist er doch die bevorzugte mediale Praktik innerhalb der Kulturtechnik Programmieren. Seit seiner klassischen Anwendung in der theologischen Exegese und in der juristischen Praxis seit der Spätantike dient der Kommentar dazu, Texte zu stabilisieren, zu kritisieren und zu rechtfertigen. Ob es sich um religiöses oder ziviles Recht handelt, der Kommentar bewahrt es davor, träge oder unverständlich zu werden; er hält die Argumente im Fluss, indem er bestimmte Annahmen, Urteile und Aussagen unterstreicht und andere, neue in den Diskurs einbringt.²¹ Eine ähnliche Funktion hat der Kommentar in der philologischen Textanalyse, bei der Erstellung kritischer Editionen und in der *critique génétique*, wie sie von Gérard Genette (1997/1992) und Almuth Grésillon (1999) entwickelt wurde.²² Kommentare in der Editionsphilologie zeigen auf, wo der Text unklar oder mehrdeutig ist, wo es Abweichungen oder Streichungen im Original gibt, um auf diese Weise die Entstehung und den Aufbau des Textes, die unterschiedlichen Varianten und Umarbeitungen transparent zu machen. Und schließlich kann die Notwendigkeit des Kommentars im Kontext der digitalen Philologie gesehen werden, die etablierte Methoden der Jurisprudenz und der Philologie auf Software überträgt.²³

²¹ Cornelia Vismann und Markus Krajewski, „Computer-Juridisms“, in: *Grey Room. Architecture, Art, Media, Politics* 8:29 (2007), S. 90–109, hier S. 102; Markus Krajewski und Cornelia Vismann, „Kommentar, Code und Kodifikation“, in: *Zeitschrift für Ideengeschichte* 3:1 (2009), S. 5–16, hier S. 5–9.

²² Vgl. Gérard Genette, *Paratexte. Das Buch vom Beiwerk des Buches*, Frankfurt a.M., New York: Campus 1997/1992 und Almuth Grésillon, *Literarische Handschriften. Einführung in die critique génétique*, Bern: Peter Lang 1999.

²³ Moritz Hiller, „Diskurs/Signal (II). Prolegomena zu einer Philologie digitaler Quelltexte“, in: *editio. Internationales Jahrbuch für Editionswissenschaft* 28 (2014), S. 192–212; Thorsten Ries, „die geräte klüger als ihre besitzer‘: Philologische Durchblicke hinter die Graphical User Interface. Überlegungen zur digitalen Quellenphilologie, Studie zu Michael Speiers ‚ausfahrt st. nazaire‘“, in: *editio. Internationales Jahrbuch für Editionswissenschaft* 24 (2010), S. 149–199; Nick Montfort, *Exploratory programming for the*

Im Rahmen der hier vorgeschlagenen Methodik der Quellcodekritik umfasst der Kommentar all diese Funktionen. Die Methode geht jedoch einen entscheidenden Schritt weiter, indem sie dem Kommentar seine gesamte epistemische Reichweite entfalten lässt. Denn eine der größten erkenntnisfördernden Eigenschaften des Kommentars liegt im systematischen Übergang zwischen den Ebenen: Zwischen Text und Kommentar klafft immer eine kleine Lücke. Das Überschreiten dieser Kluft bewirkt einen automatischen Perspektivwechsel, verbunden mit dem notwendigerweise distanzierenden Blick von der Ebene des Kommentars auf den Gegenstand der Analyse, sei es ein literarischer Text oder eben der Teil eines Algorithmus. Dieser Perspektivwechsel beinhaltet einen Moment der Selbstreflexion, in dem der Kommentar eine kritische Auseinandersetzung mit dem Prozess des Schreibens hervorruft. Indem er sich systematisch zwischen den Ebenen bewegt, schafft der Kommentar einen epistemischen Hebel, der eine ständig oszillierende Perspektive auf das Geschriebene ermöglicht und als solcher ein bis dato unzureichend genutztes epistemologisches Instrument darstellt. Denn der Kommentar dient stets als unaufdringlicher Assistent der Reflexion, der zur Erklärung und Plausibilisierung, zur Exegese und zur Verknüpfung mit anderen Texten einlädt und provoziert. Das Konzept der Quellcodekritik nutzt diese oszillierende Perspektive zwischen operativen und erklärenden Segmenten in einer Datei, um den Kommentar des Codes zum eigentlichen Text zu machen.

Die Methode der Quellcodekritik umfasst also zweierlei: Auf der pragmatischen Ebene heißt das erstens, dass Software nicht nur ausgeführt, sondern mit Selbstverständlichkeit stets einer systematischen – oder, wie Till Heilmann in seinem Beitrag vorführt: kursorischen – kritischen Lektüre unterzogen wird,

arts and humanities, Cambridge, MA: The MIT Press 2016, sowie die digitalen Archivierungsprojekte am Deutschen Literaturarchiv Marbach, z.B. die Kuratierung der digitalen Quellen Friedrich Kittlers.

die ihre Algorithmen durch Erklärungen, Reflexionen, Verweise und womöglich auch durch Modifikationen des Codes verständlich und plausibel macht. Das Ziel dieser Methode ist dabei nicht nur Transparenz und Verständlichkeit um der Didaktik willen, sondern mehr noch ein tiefer gehendes Verständnis der Programmstrukturen zur Verbesserung einer grundsätzlichen *coding literacy*. Zweitens zielt die Methode auf der epistemischen Ebene, die wiederum über die von Knuth angestrebten Effekte des *Literate Programming*-Prinzips hinausgeht, darauf ab, durch ausführliche Kommentierung den Code zu narrativieren, zu historisieren und zu diskursivieren. Kurz gesagt: Der fortlaufende Kommentar innerhalb des Codes soll *zum eigentlichen Text werden*, was nichts anderes bedeutet, als den Code wie eine Geschichte zu schreiben (vgl. Listing 3). Ausblickend lässt sich hoffen, dass dieser Ansatz die gut etablierten Verfahren der historischen Quellenkritik in die Produktionsstätten der digitalen Gegenwart, in die Repositorien und IDEs künftiger Softwareprojekte portiert und damit nichts Geringeres anstrebt, als einen neuen Standard für das Schreiben von Code in den Geisteswissenschaften zu setzen vermag.

Demzufolge bietet ein solches Verständnis von Code ein nicht unwesentliches emanzipatorisches Potenzial. Quellcodekritik bedeutet, *coding literacy* auf eine Ebene zu heben, die eine Reflexion über Machtstrukturen in digitalen Gesellschaften ermöglicht. Das Programmieren als Kulturtechnik befähigt als eine reflektierende Praktik des Programmierens demnach dazu, Code zu verstehen, einzuordnen und mithin auch zu schreiben, um angesichts der aktuellen Macht der Informationstechnologien eine bestimmte Handlungsfähigkeit zurückzugewinnen.

Listing 3: A Very Brief Example of Source Code Criticism

```

439  /* How to get surprised by your digital assistant?
440  Computers don't do well with true randomness (Rubin 2011-01.11.2011).
441  Therefore, the machine, for example, can't provide a truly random card
442  from a collection of index cards. Below, we explore four ways in which
443  surprise can be implemented, based on historic examples of designing
444  serendipity...*/
445
446  public int getRandomInt(int max) {
447      /* This method selects a pseudo-random card from the card index
448      collection.
449      It takes 'int max' as input, indicating the collection's current amount of
450      cards. */
451      // The number to be determined will be chosen from a range between
452      max and this minimal value
453      int min = 1;
454      // JAVA can now create a pseudo-random integer by this new object:
455      Random randomGenerator = new Random(0);
456      // With the result of randomGenerator this method then returns an
457      'accidentally' chosen number between 1 and the max amount of cards:
458      return randomGenerator.nextInt(max - min) + min;
459  }
460
461  // Inform the user directly about Mallarmé (1897) and his coup de dés:
462  System.out.println("The dice roll ended at " +
463  getRandomInt(JSynapsen.recAnz));
464
465  /* Implementing serendipity with Aby Warburg
466  Since serendipity is even more difficult to implement and at the same time
467  still a young field in information science (McCay-Peet und Toms 2017), we
468  create another unfolding opportunity for >controlled coincidence< by
469  bringing Aby Warburg's >law of the good neighbour< to bear (see Krajewski
470  2017, 99-101): Since 1924, library staff Gertrude Bing and Fritz Saxl, in the
471  Kulturwissenschaftlichen Bibliothek Warburg (K.W.B.) in Hamburg and
472  later in London, set aside similar texts and corresponding thoughts for a
473  given book by constantly rearranging the shelves, thus creating new spatial
474  connections for a book, transforming the previously remote into propinquity
475  through establishing short distances. This creates new links between
476  heterogeneous units of text in close proximity, which can be explored by
477  the reader with the confidence that a knowledgeable hand has curated the
478  surrounding texts half by chance, half intentionally. We simulate this
479  knowledgeable hand by computing similarities between texts in three
480  different ways:
481  1. by the number of matching terms with which the texts are tagged. The
482  highest match leads to the greatest spatial proximity.
483  2. by matching the full texts. The entire content of a book is examined with
484  all other full texts for word matches and assigned a characteristic value.
485  Again, the texts with the highest similarities move close together.
486  3. by an analysis of the quotations in the book. The texts that are cited in
487  the selected book and are also present in the local database are included in
488  the neighborhood, the highest citation frequency determines the closest
489  proximity.a
490  The respective neighborhoods of the selected text can then be illustrated in
491  the form of a sociograph in the diagram. */
492
493  public String[] generateGoodNeighboursByHeadwords(string keyword) {
494      /* First we retrieve a list of headwords from the database associated
495      with this card, identified by its keyword like >warburg:1924< [...]
496      This method returns a String array with a list of keyword,
497      representing the neighbourhood of the chosen card. */
498      System.out.println("... This code continues...");
499  }
500
501  a. There are, of course, more alternatives.
502  b. There are, of course, more alternatives.

```

Listing 3:
A Very Brief Example of Source Code Criticism

WIE LIEST MAN 100.000 ZEILEN CODE?

Till A. Heilmann

I.

In den Critical Code Studies (CCS) sind meist recht kleine Programme oder kurze Ausschnitte aus längeren Quelltexten Gegenstand, manchmal gar nur eine einzelne Zeile Code.¹ Die Gründe sind leicht ersichtlich. Computerprogramme sind in formalen, nicht in sogenannten natürlichen Sprachen geschrieben. Auch ist ihr Aufbau mit Kontrollstrukturen wie Verzweigungen und Schleifen oder mit objektorientierter Modellierung in aller Regel nicht linear zu erfassen. Bereits bei wenigen Hundert Zeilen verliert man schnell den Überblick. Das Problem verschärft sich noch, wenn bei der Autor:innen- wie der Leser:innenschaft nur rudimentäre oder überhaupt keine praktischen Programmierkenntnisse vorhanden sind. Es überrascht daher nicht, dass viele Analysen in den CCS oft nicht kommerzielle oder nicht professionelle kleine Software betreffen, nicht selten auch künstlerische Projekte und damit Code, der wenigstens teilweise geschrieben wurde, um als Code gelesen (und nicht nur ausgeführt) zu werden.²

Wollen die CCS aber die kulturelle Prägung und Wirksamkeit von Software im gesellschaftlichen Maßstab untersuchen, dann haben sie sich mit Code auseinanderzusetzen, der in den entsprechenden Größenordnungen produktiv ist. Es geht dann

¹ Siehe beispielhaft die Analysen von Mark C. Marino, *Critical Code Studies*, Cambridge, MA: MIT Press 2020; Ea Christina Willumsen, „Source Code and Formal Analysis: A Reading of Passage“, in: *Transactions of the Digital Games Research Association* 3:2 (2017), S. 213–235; Nick Montfort u.a., *10 PRINT CHR\$(205,5)+RND(1)); : GOTO 10*, Cambridge, MA: MIT Press 2012, <http://10print.org> [Letzter Zugriff: 15.8.2023].

² Siehe etwa die *case studies* in Mark C. Marino, „Reading Culture Through Code“, in: Jentery Sayers (Hg.), *Routledge Companion to Media Studies and Digital Humanities*, London: Routledge 2018, S. 472–483.

um Standardanwendungen und Apps, AAA-Spiele, Betriebssysteme für Mobilgeräte, PCs und Server oder ganze Softwarestacks, also um Textverarbeitungsprogramme wie Microsoft Word, Social Media-Clients wie Instagram, Spiele wie Minecraft oder Entwickler-Frameworks wie die Unreal Engine und Gesamtlösungen wie LAMP (Linux, Apache, MySQL und PHP).³ Für solche Software wird der Umfang heute in Millionen Zeilen Code gemessen. Die Quelltexte, auf denen Googles Chrome-Browser basiert, zählten im Sommer 2023 knapp 30 Millionen Codezeilen in nicht weniger als 40 verschiedenen Programmiersprachen.⁴

Zum Glück oder Unglück für die CCS stellt sich die Länge von Code jedoch häufig gar nicht erst als Problem. Denn der Quelltext kommerzieller Software ist meist als Firmengeheimnis geschützt und für Dritte nicht zugänglich. Kritische Codelektüren müssen sich deshalb auf Free- und Open-Source-Software beschränken oder mit Programmen begnügen, deren Code von den jeweiligen Firmen für die Öffentlichkeit freigegeben wurde. Eine schöne Sammlung solcher Software hält etwa das Computer History Museum (CHM) auf seiner Webseite bereit.⁵ Dort findet man unter anderem Quelltexte einiger wichtiger Programme aus den ersten zwei Jahrzehnten der PC-Geschichte, so von Apple MacPaint 1.3 (1984) oder Microsoft Word für Windows 1.1a (1991) – und auch von Adobe Photoshop 1.0.1 (1990), das uns hier als Testfall für die CCS dienen soll.

Im Vergleich zu aktueller Software muten Programme der Vergangenheit geradezu winzig an. Doch ist auch der Quelltext von Photoshop 1.0.1 mehr als hunderttausend Zeilen lang, „128,000 lines of mostly uncommented but well-structured

³ Mindestens so wichtig wäre die Untersuchung von Software eingebetteter Systeme wie Fitness-Trackern, Autos oder Smart-Home-Netzwerken.

⁴ Siehe https://openhub.net/p/chrome/analyses/latest/languages_summary [Letzter Zugriff: 15.8.2023].

⁵ Siehe <https://computerhistory.org/playlists/source-code/> [Letzter Zugriff: 15.8.2023].

code“.⁶ Natürlich hinkt der Vergleich, aber legt man für eine typische Buchseite 30 Zeilen Text zugrunde, dann entspricht der Umfang der genannten Version von Photoshop in Codezeilen mehr als 3000 Buchseiten. Das kommt in die Nähe von Prousts Hauptwerk. Und man liest Computerprogramme eben nicht wie Romane. Die schiere Länge, die Komplexität der Architektur und die besondere Sprache – um nur die wichtigsten Faktoren zu nennen – machen die Lektüre von Programmen, wie sie die CCS propagieren, schon für den Fall jahrzehntelanger Anwendungssoftware zur Herausforderung. An Photoshop 1.0.1 lässt sich gut vorführen, mit welchen Schwierigkeiten man konfrontiert wird, wenn man Quelltexte dieser Größe ohne Informatikausbildung und mit nur geringen Programmierfähigkeiten lesen will. Die Frage, *weshalb* man in den Geistes- und Kulturwissenschaften Quelltexte lesen sollte, bleibt dabei allerdings ausgeklammert.⁷ Es wird hier allein um das *Wie* gehen. Wie also liest man hunderttausend Zeilen Code?

Um die Antwort vorwegzunehmen: oberflächlich. Wenigstens für den Einstieg in die Analyse großer Programme bieten sich summarische, quantitativ-statistische Lektüreverfahren an, gewissermaßen ein auf einen einzelnen Programmtext bezogenes *distant reading*.⁸ Das ist nicht nur der Menge des Codes geschuldet, sondern ebenso sehr den Autor:innen und Leser:innen seiner Analyse. Selbstverständlich verlangen die CCS ein Mindestmaß an *code literacy*. Wie herkömmliche Philologien ohne Lese- und Schreibfähigkeit undenkbar sind, so ist eine Kritik von Quelltexten ohne basales Wissen von Com-

⁶ Siehe Len Shustek, „Adobe Photoshop Source Code“, Computer History Museum, Februar 2012, <https://computerhistory.org/blog/adobe-photoshop-source-code/> [Letzter Zugriff: 15.12.2019].

⁷ Siehe dazu Markus Krajewski, „Against the Power of Algorithms Closing, Literate Programming, and Source Code Critique“, in: *Law Text Culture* 23 (2019), S. 119–133; Marino, *Critical Code Studies*, S. 37–54.

⁸ Siehe Franco Moretti, „Style, Inc.: Reflections on Seven Thousand Titles (British Novels, 1740–1850)“, in: *Critical Inquiry* 36:1 (2009), S. 134–158.

puterprogrammierung eine Unmöglichkeit. Aber wie wir von Mediävist:innen und Altorientalist:innen erwarten, das Rolandslied oder das Gilgamesch-Epos lesen zu können, ohne von ihnen selbst literarische Werke vergleichbarer Qualität zu fordern, so ist es nicht Ziel der CCS, das nächste Photoshop oder einen besseren Webbrowser zu schreiben. Grundkenntnisse von Algorithmen und Datenstrukturen sowie der untersuchten Sprachen und erste Programmiererfahrungen darin sollten ausreichen. Da das expressive Potential von Programmiersprachen sehr viel kleiner als das sogenannter natürlicher Sprachen ist, dürften schon geringe Lese- und Schreibfähigkeiten genügen – vergleichbar vielleicht der Stufe A2 oder B1 nach dem Gemeinsamen europäischen Referenzrahmen für Sprachen.

Bernhard Rieder hat argumentiert, dass es nicht damit getan sei, Code zu lesen, wenn man ein Computerprogramm richtig verstehen will, da der Geltungsbereich seiner Funktion die Komplexität von Code bei Weitem übersteige.⁹ Trifft diese Mahnung zu, dann ist das für die CCS eine gute Nachricht. Denn worauf diese hinauswollen, liegt ohnehin jenseits des Codes. So lesen wir den Quelltext von Adobe Photoshop, um schließlich etwas über den medialen Status von Bildern in digitalen Formaten, über die Veränderung fotografischer Praktiken und die Transformation visueller Kulturen zu erfahren. Das sind keine Sachverhalte, die sich allein auf technische Funktionen oder den Quelltext eines Programms reduzieren ließen. Der Code soll vielmehr als informatives Fenster dienen, durch das hindurch wir Ausblicke auf die digitale Verfasstheit unserer Gegenwart erhalten. Entsprechend ist der exakte Nachvollzug eines Programms auf Ebene seines Codes in allen Einzelheiten auch nicht nötig.

⁹ Vgl. Bernhard Rieder, *Engines of Order: A Mechanology of Algorithmic Techniques*, Amsterdam: Amsterdam University Press 2020, S. 14 und 98.

Ziel des hier vorgeschlagenen *distant reading* von Quelltexten ist es, große Mengen Code erst einmal so in den Griff zu bekommen, dass sie für nachfolgende Analysen einzelner Programmteile und für die Interpretation kultureller Prägungen und Auswirkungen der Software überhaupt zugänglich werden. Diese Aufbereitung geschieht vor allem durch *Reduktion*. Wir werden den Quelltext von Photoshop hinsichtlich verschiedener Aspekte maschinell lesen, wobei als Resultat unserer maschinellen Lektüren schritt- und stückweise neue, kleinere Texte entstehen, die ihrerseits wieder und anders gelesen werden können. Im Vordergrund steht, die Anlage des Codes in seinen Grundzügen zu überschauen, maßgebliche Elemente (wie häufig verwendete Variablen, Funktionen u.Ä.) auszumachen, Muster und Regelmäßigkeiten zu erkennen oder, gerade umgekehrt, auf ungewöhnliche Konstrukte und Entscheidungen aufmerksam zu werden.

Das maschinelle Lesen und Schreiben des Codes erfolgt mit üblichen, frei verfügbaren Unix-Werkzeugen für die Kommandozeile, die für die Umformung und Auswertung von Zeichenketten – also letztlich zur Verarbeitung von Text – gemacht sind. Die meisten dieser Werkzeuge sind auf jedem Unix-System und macOS-Computer bereits installiert, können aber auch auf Windows-Rechnern nachgerüstet werden.¹⁰ Sie sind gerade in der Kombination mächtig, aber im einfachen Gebrauch schnell zu erlernen.¹¹

Bedauerlicherweise gibt es in den CCS kein Beispiel dafür, wie mit Programmen vom Umfang gängiger Anwendungssoftware zu verfahren ist. Meines Wissens existiert bislang kein

¹⁰ Manche mit macOS ausgelieferte Unix-Tools wie `sed` oder `awk` sind ältere Versionen der Programme, die u.U. nicht richtig funktionieren. Im Folgenden werden stets aktuelle Versionen des GNU-Projekts verwendet. Zu Windows-Rechnern siehe das Cygwin-Projekt auf <https://cygwin.com/> [Letzter Zugriff: 15.8.2023].

¹¹ Siehe bspw. Brad Yoes, „Introduction to Text Manipulation on UNIX-Based Systems: Using Standard Utilities“, März 2012, <https://developer.ibm.com/articles/au-unixtext/> [Letzter Zugriff: 15.8.2023].

methodisches Gerüst für die geistes- und kulturwissenschaftliche Analyse und Interpretation von Code-Mengen dieser Größenordnung. Was folgt, ist daher ein Werkstattbericht aus meiner laufenden Softwarestudie zu Photoshop:¹² ein erster Anlauf, ein tastender Versuch, dem Quelltext des Programms ohne viele Vorkenntnisse oder im Voraus festgelegten Plan etwas zu entlocken und ihn für weitergehende Analysen aufzubereiten. Wir beginnen als naive Leser:innen mit einfachen Beobachtungen an den ‚Oberflächen‘ des Codes, vertiefen unsere Lektüre nach und nach und sehen, wohin uns die maschinelle Reduktion des Quelltextes und unsere Schlussfolgerungen dazu führen.

Wie schon erwähnt, setzen wir nicht mehr voraus als grundlegende Computerkenntnisse, ein einfaches Verständnis davon, wie Programmierung funktioniert, sowie ein wenig Erfahrung im Umgang mit Unix-Tools und dem Schreiben kurzer Skripte auf der Kommandozeile. Wir vertrauen zudem auf das Internet: Eine gute Stichwortsuche wird uns hoffentlich, falls nötig, zu hilfreichen Webseiten führen; und von Wikipedia erwarten wir, wenigstens in englischer Sprache, gut belegte Artikel zu Fragen der Programmierung.

2.

Die erste Schwierigkeit, die sich bei der Lektüre großer Programme stellt, ist die, wo man zu lesen beginnt. Welche von den zehn- oder hunderttausend Zeilen Code ist die richtige? Theoretisch könnte man an jedem beliebigen Punkt im Quelltext ansetzen (bei der Deklaration einer bestimmten Funktion etwa) oder, gerade andersherum, versuchen, den richtigen ‚Anfang‘ des Programms zu finden und von da fortzufahren.

¹² Siehe Till A. Heilmann, „Processed Pictures, Photoshop, and Unsharp Mask“, in: *International Journal for Digital Art History* 6 (2021), S. 169–177.

Beide Ansätze sind jedoch problematisch. Die Länge und der Aufbau großer Programme verbieten ein lineares, gründliches Lesen von Code nach dem Vorbild intensiver Romanlektüre; andererseits machen sie ein Herausgreifen einzelner Teile aus dem Quelltext schwierig, weil anfangs unklar ist, welche Stellen wofür relevant sind. Weder lässt sich umfangreiche Software gleichsam von der ersten bis zur letzten Zeile durchlesen noch in ihrem Zusammenhang von verstreuten Ausschnitten her aufschließen. Nötig ist deshalb ein Panorama des Quelltextes, das einen ersten Gesamteindruck vermittelt und die Planung der nächsten Schritte erlaubt.

Beginnen wir unsere Untersuchung da, wo uns der Code von Adobe Photoshop zur Verfügung gestellt wird: Hat man auf der Webseite des CHM der entsprechenden Lizenzvereinbarung zugestimmt,¹³ darf man eine ZIP-Datei mit dem Titel `photoshop-v.1.0.1-source-code.zip` herunterladen. Entpackt man dieses rund 600 Kilobyte große Archiv, findet man darin nicht einen einzigen Quelltext, sondern 179 Dateien mit Namen von `About.r` bis `UVMemory.p`. Das ist nicht erstaunlich, wird doch der Code von Programmen (auch von kleineren) aus Gründen der Übersichtlichkeit und modularen Verwendbarkeit meist in mehrere, möglichst kurze Teile aufgespaltet. Zudem – die Dateinamenserweiterungen deuten es hier bereits an – sind Programme häufig aus unterschiedlichen Sprachen und Datensätzen zusammengefügt. Wir kommen darauf zurück.

Noch vor den Inhalten der Dateien sagen allein ihre Namen und Größen etwas über den Aufbau des Quelltextes von Photoshop 1.0.1 aus. Denn viele Dateinamen geben zumindest Hinweise darauf, welche Aufgaben erfüllt werden. So scheinen manche Dateien direkt einzelnen Funktionen des Programms zu korrespondieren, etwa dem Zuschneiden, Verkleinern und Vergrößern oder Drehen von Bildern (`UCrop.*`, `UResize.*` und

¹³ Siehe Shustek, „Adobe Photoshop Source Code“.

URotate.*). Andere implementieren wohl die unterstützten Bildformate wie EPS, GIF oder TIFF (UEPSFormat.*, UGIFFormat.* und UTIFFFormat.*). Wieder andere beziehen sich vermutlich auf die Algorithmen und Verfahren, mit denen Bilder verarbeitet und gespeichert werden (ULZWCompress.* oder UVMemory.*). Schließlich gibt es fünf Dateien, die nach dem Programm selbst benannt sind (MPhotoshop.p, Photoshop.make, Photoshop.r, UPhotoshop.incl.p, UPhotoshop.p). Nur ganz wenige Dateien sind so unbestimmt oder undurchsichtig benannt, dass sich vorerst kaum etwas daraus ableiten lässt (Black.r, Tables.* oder UGhost.*). Was zusätzlich auffällt, ist die Vielzahl der Dateinamen, die mit einem großen U beginnen, etwa UCrop.*, UGIFFormat.* oder UVMemory.* – es sind, um genau zu sein, 156. Der Grund dafür wird sich vielleicht im weiteren Verlauf unserer Lektüre erweisen.

Der Quelltext von Adobe Photoshop ist also mutmaßlich entlang der Grenzen von Programmfunktionen, Schnittstellen und Formaten sowie verschiedener interner Mechanismen in einzelne Textstücke aufgeteilt. Diese Strukturierung des Codes ist nicht nur Kennzeichen eines der beiden großen Programmierparadigmen, denen der Entwurf von Photoshop folgt (und das, wenig überraschend, strukturierte Programmierung heißt);¹⁴ sie könnte sich auch als hilfreich erweisen, insofern sie die thematische Suche nach bestimmten Codeteilen später erleichtert.

Nehmen wir als Nächstes die Endungen der Dateinamen in Augenschein. Jede Datei trägt eine Erweiterung wie .p oder .r, die traditionell den Typ der Datei angibt. Auf der Webseite des CHM haben wir bereits erfahren, dass die erste Version von Photoshop hauptsächlich in der Programmiersprache Pascal geschrieben wurde, mit einigen zusätzlichen Teilen in Assem-

¹⁴ Siehe https://de.wikipedia.org/wiki/Strukturierte_Programmierung [Letzter Zugriff: 16.8.2023].

blersprache für den Motorola 68000-Prozessor damaliger Macintosh-Computer.¹⁵ Wir können also schlussfolgern, dass das Kürzel `.p` für Pascal und das Kürzel `.a` für Assemblersprache steht. Eine schnelle Zählung – im Dateimanager oder auf der Kommandozeile mit `$ ls *.p|wc -l; ls *.a|wc -l` – ergibt ein Verhältnis von 94 zu 21 Dateien vom Typ `.p` und `.a`, was die Vermutung stützt.¹⁶ An diesem Punkt unserer Lektüre ‚öffnen‘ wir zum ersten Mal den Quelltext von Photoshop und schauen, um letzte Zweifel bezüglich der Dateitypen auszuräumen, probierhalber mit einem beliebigen Texteditor in zwei ausgewählte Teile des Quelltextes hinein: die mit verschiedenen Endungen versehenen, ansonsten aber namensgleichen Dateien `URotate.p` und `URotate.a`. Zuerst ein kurzer Auszug aus `URotate.p`, Zeilen 1602–1645 (ohne Leerzeilen):

```
1602 FUNCTION DoRotateCommand (view: TImageView; angle: INTEGER): TCommand;
1604     VAR
1605         doc: TImageDocument;
1606         aRotateImageCommand: TRotateImageCommand;
1607         aRotateFloatCommand: TRotateFloatCommand;
1609     BEGIN
1611     IF angle = 1800 THEN
1612         DoRotateCommand := DoFlipCommand (view, TRUE, TRUE)
1614     ELSE
1615         BEGIN
1617         doc := TImageDocument (view.fDocument);
1619         IF EmptyRect (doc.fSelectionRect) THEN
1620             BEGIN
1622                 NEW (aRotateImageCommand);
```

¹⁵ Siehe Shustek, „Adobe Photoshop Source Code“.

¹⁶ Das Dollarzeichen `$` meint hier und andernorts den Prompt der Kommandozeile, auf den die eingegebenen Befehle – hier also `ls *.p|wc -l; ls *.a|wc -l` – folgen. Tatsächlich gehören, wie wir gleich sehen werden, auch Dateien mit der Endung `.inc` zur Sprache Pascal, weshalb das eigentliche Verhältnis bei 144 zu 21 liegt.

```

1623         FailNil (aRotateImageCommand);
1625         aRotateImageCommand.IRotateImageCommand (view, angle);
1627         DoRotateCommand := aRotateImageCommand
1629         END
1631     ELSE
1632         BEGIN
1634         NEW (aRotateFloatCommand);
1635         FailNil (aRotateFloatCommand);
1637         aRotateFloatCommand.IRotateFloatCommand (view, angle);
1639         DoRotateCommand := aRotateFloatCommand
1641         END
1643     END
1645 END;17

```

Und nun ein Auszug aus URotate.a, Zeilen 12–57 (wieder ohne Leerzeilen):

```

12             SEG         'ADoRotate'
14 DoReverseBytes PROC     EXPORT
16 ;           Calling sequence (Pascal conventions):
17 ;
18 ;           PROCEDURE DoReverseBytes (data: Ptr;
19 ;                                     count: INTEGER);
20 ;
21 ;           Parameter Offsets
23 @data      EQU        10
24 @count     EQU        8
26 ;           Size of parameters
28 @params    EQU        6
30 ;           Unload parameters
32           LINK        A6,#0
33           MOVE.L     @data(A6),A0
34           MOVE.W     @count(A6),D0

```

¹⁷ Die fortlaufenden Zahlen zu Beginn jeder Zeile hier und andernorts sind Zeilennummern und jeweils *nicht* Teil des Codes.


```
36 ;          Compute address of one pixel past last
38          MOVE.L    A0,A1
39          ADDA.W    D0,A1
41 ;          Reverse the bytes
43          ASR.W     #1,D0
44          SUB.W     #1,D0
45          BMI.S     @2
47 @1        MOVE.B   (A0),D1
48          MOVE.B   -(A1),(A0)+
49          MOVE.B   D1,(A1)
50          DBF      D0,@1
52 ;          Clean up and exit
54 @2        UNLK     A6
55          MOVE.L   (SP)+,A0
56          ADD.W    #@params,SP
57          JMP     (A0)
```

Die Unterschiede zwischen den beiden Texten stechen ins Auge, auch wenn man keine tieferen Programmierkenntnisse besitzt.¹⁸ Der erste Ausschnitt zeigt gut zwei wesentliche Merkmale einer höheren Programmiersprache wie Pascal, nämlich abstrakte Prozeduren und Variablen mit deskriptiven Namen (wie `DoRotateCommand` oder `angle`) sowie logische Kontrollstrukturen (wie `IF ... THEN ... ELSE`). Im zweiten Ausschnitt fallen dagegen die für eine Assemblersprache typische Litanei sich wiederholender Kürzel (wie `MOVE.*`, `ADD.*` und `SUB.*`) sowie die Adressierung einzelner Register des Motorola 68000-Prozessors (wie `A0` und `D0`) auf. Die Differenz zwischen der höheren Sprache einerseits und der maschinen-nahen Programmierung andererseits wird auch darin sichtbar,

¹⁸ Eine brauchbare Kurzdarstellung von Pascal und der Assemblersprache für Motorola 68000-Prozessoren gibt Wikipedia auf [https://de.wikipedia.org/wiki/Pascal_\(Programmiersprache\)#Pascal_und_Mac_OS](https://de.wikipedia.org/wiki/Pascal_(Programmiersprache)#Pascal_und_Mac_OS) und https://de.wikipedia.org/wiki/Motorola_68000 [Letzte Zugriffe: 16.8.2023].

dass der Code für Assembler mehrere Kommentare zur Erläuterung des Geschehens enthält (in Zeilen, die mit einem Semikolon beginnen, z.B. `Compute address of one pixel past last`); dagegen scheint der in Pascal geschriebene Teil so selbstklärend, dass er keiner weiteren Kommentierung bedarf.

Sehen wir uns nun die Endungen der restlichen Dateien an. Es sind dies `.inc`, `.make`, `.r`, `.t` und `.txt`. Zwar gibt es auf Wikipedia eine lange Liste, die viele Dateinamenserweiterungen aufschlüsselt,¹⁹ für unseren Fall ist sie allerdings wenig hilfreich. Während die Endung `.txt` wohlbekannt sein dürfte (und hier bloß in der Datei `ChangeHistory.txt` auftaucht), findet man sonst nur zu `.inc` den knappen Hinweis: „Pascal, Borland Delphi, C, C++ SourceCode zum mehrfachen Wiederverwendung [sic] in Sourcecode-Dateien“.²⁰ Wir sind also auf uns allein gestellt.

Zuerst die `.inc`-Dateien: Sie sind im Vergleich zu den anderen Dateien allesamt kurz oder sehr kurz. Der Dateimanager oder das Unix-Tool `ls` mit dem entsprechenden Suchmuster und der Option `S` für Sortierung nach Größe der Dateien (`$ ls *.inc -S`) zeigen zwischen 355 Byte und 4,3 Kilobyte an. Pascal-Dateien hingegen sind zwischen 732 Byte und 210 Kilobyte groß (`$ ls *.p -S`). Sieht man sich die Inhalte der `.inc`-Dateien an, dann stellt man fest, dass darin ausschließlich Verweise auf Funktionen und Prozeduren enthalten sind, die in den Pascal-Dateien definiert werden – also etwa in `URotate.p.inc`, Zeile 18, die bereits erwähnte Funktion `DoRotateCommand` aus `URotate.p`, Zeilen 1602–1645. Dateien vom Typ `.inc` können für die weitere Analyse also als Pascal-Dateien gezählt und, was ihren Inhalt betrifft, im Grunde ignoriert werden.

Von den drei `.make`-Dateien scheint `Photoshop.make` die wichtigste zu sein. Jedenfalls ist sie mit die 9,4 Kilobyte die

¹⁹ Siehe https://de.wikipedia.org/wiki/Liste_von_Dateinamenserweiterungen [Letzter Zugriff: 15.6.2023].

²⁰ Ebd.

größte. Vor allem aber führt sie die übrigen Dateien des Quelltextes zusammen, wie etwa in den Zeilen 71–82:

```
71 UPhotoshop.p.o : UConstants.p UVMemory.p UBWDialoq.p UProgress.p \  
72     UAbout.p.inc UAdjust.p.inc UAssembly.a.inc \  
73     UCalculate.p.inc UChannel.p.inc \  
74     UConvert.p.inc UCoords.p.inc UCrop.p.inc \  
75     UDither.a.inc UDraw.p.inc UFilter.p.inc UFloat.p.inc UGhost.p.inc \  
76     UHistogram.p.inc UInitFormats.p.inc UInternal.p.inc \  
77     ULine.p.inc UMagnification.p.inc \  
78     UPasteControls.p.inc UPick.p.inc UPreferences.p.inc \  
79     UPrint.p.inc UResize.p.inc \  
80     URotate.p.inc UScan.p.inc UScreen.a.inc UScreen.p.inc \  
81     USelect.p.inc USeparation.p.inc UTable.p.inc UText.p.inc \  
82     UTrap.p.inc
```

Photoshop.make ist anscheinend ein Makefile:²¹ eine Hilfsdatei, die dem Compiler sagt, wie die Teile des Quelltextes zueinander in Beziehung stehen und wie der aus den einzelnen Pascal- und Assembler-Dateien übersetzte Objektcode zur fertigen, lauffähigen Anwendung für Enduser:innen zusammengefügt wird. In den Zeilen 31–33 erfahren wir außerdem, dass die uns bislang unbekannt Dateitypen mit der Endung `.r` sogenannte „RezFiles“ sind.

```
31 OtherRezFiles = "{RIncludes}"SysTypes.r \  
32     About.r Black.r Huffman1.r Huffman2.r \  
33     PixelPaint.r Tables.r Tips.r
```

Schauen wir versuchsweise ins RezFile `About.r`, dann sehen wir zwei lange Blöcke von Daten, die hexadezimal codiert und

²¹ Siehe <https://de.wikipedia.org/wiki/Make> [Letzter Zugriff: 16.8.2023].

jeweils als resource 'PICT' überschrieben sind, beginnend in Zeile 8:

```

08 resource 'PICT' (700, purgeable)
09   {
10     18908,
11     {0, 0, 220, 500},
12     "$0011 02FF 0C00 0000 49DC 0000 0000 0000"
13     "$0000 00DC 0000 01F4 0000 0000 0000 0001"
14     "$000A 0000 0000 00DC 01F4 0098 80FA 0000"
15     "$0000 00DC 01F4 0000 0000 0000 0000 0048"
16     "$0000 0048 0000 0000 0004 0001 0004 0000"
17     "$0000 0000 0000 0000 0000 0000 0000 0000"

```

Es ist zu vermuten, dass hier digitale Bestände (*resource[s]*) von Bildern (*PICT[ures]*) abgelegt sind – in diesem Fall höchstwahrscheinlich die Bitmaps für die Grafiken in der Dialogbox „About Photoshop...“, die über den entsprechenden Menüpunkt im Programm angezeigt wird (Abb. 1). Die übrigen RezFiles enthalten weitere Datenbestände auch anderer Typen als PICT und in anderen Codierungen. Sie sollen hier nicht weiter verfolgt werden; aber ein flüchtiger Blick etwa in `Tips.r` legt nahe, dass dort die Formen der in Photoshop verfügbaren digitalen Pinselspitzen gespeichert sind, und die sehr lange Datei `Photoshop.r` enthält Hunderte von Datensätzen ganz unterschiedlicher Art.

Bleiben die beiden Dateien vom Typ `.t`: `Huffman1.t` und `Huffman2.t`. Sie bestehen aus langen Reihen von Dezimalzahlen, denen Binärzahlen von variabler Länge zugeordnet sind. Wie schon der Name der Dateien verrät, handelt es sich hier um Tabellen (*t[ables]*) Huffman-codierter Zahlen, die wohl der verlustfreien Kompression von Bilddaten dienen.²²

²² Siehe <https://de.wikipedia.org/wiki/Huffman-Kodierung> [Letzter Zugriff: 16.8.2023].

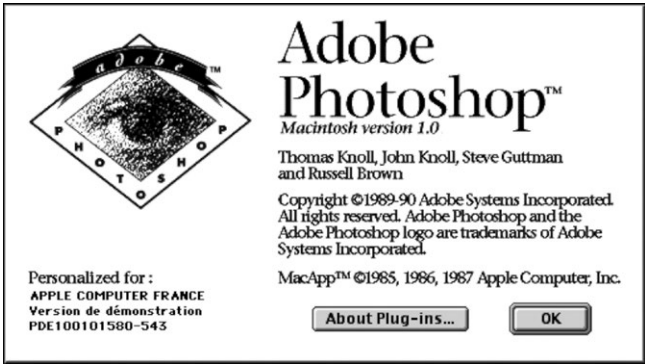


Abb. 1:
„About Photoshop...“-Dialogbox von
Adobe Photoshop 1.0, 1990 (Screenshot des Autors)

Fassen wir die Ergebnisse unserer ersten oberflächlichen Betrachtung des Codes von Adobe Photoshop 1.0.1 zusammen: Dateien vom Typ `.a`, `.p` und `.inc` enthalten die eigentlichen Anweisungen des Quelltextes in Pascal und Assemblersprache; Dateien vom Typ `.r` und `.t` speichern Daten für Bilder und anderes; Dateien vom Typ `.make` enthalten Angaben für den Compiler, wie die Teile des Quelltextes zusammenpassen.

3.

Schauen wir uns den Code nun hinsichtlich des Umfangs seiner einzelnen Bestandteile an. Um einen ungefähren Eindruck zu gewinnen, vergleichen wir erst die Dateien selbst. Von den zehn größten (`$ 1s -1S`) sind sieben Pascal-Dateien, zwei Rez-Files und eine in Assembler. Die größte Datei insgesamt ist `Photoshop.r` mit 218 Kilobyte, die uns bereits bei der Untersuchung der Dateitypen aufgefallen ist und die voller Datensätze

verschiedener Art steckt. Die größte – und einzige – Assembler-Datei unter den Top Ten ist `USeparation.a` mit rund 67 Kilobyte. Sie ist, wie die Namen der darin definierten Prozeduren `DoSeparateColors` u.a. verraten, für die Berechnung von Farbwerten in Farbräumen wie RGB und CMYK zuständig. Ihre außergewöhnliche Größe ist möglicherweise ein Hinweis darauf, dass solche Berechnungen eine speziell aufwendige Aufgabe der digitalen Bildprozessierung darstellen. Die kleinsten Dateien hingegen sind, das haben wir bereits zu Beginn festgestellt, die `.inc`-Dateien, die jeweils nur wenige Hundert Byte kurz sind (`$ ls -lrS`). Alle Pascal-Dateien zusammen umfassen gut 2 Megabyte (`$ du -ch *.p *.inc`), die Assembler-Dateien rund 388 Kilobyte (`$ du -ch *.a`), die `RezFile`- und Tabellen-Dateien zusammen rund 384 Kilobyte (`$ du -ch *.r *.t`). Das Verhältnis von Pascal zu Assemblersprache zu Datensätzen beträgt damit ungefähr 5:1:1.

Nun sind Dateigrößen zwar ein brauchbarer Indikator für Codemengen. Aber das in der Informatik gängige Maß für die Größe von Softwareprojekten ist die Anzahl von Zeilen oder *lines of code*, kurz: LOC. Einfach gesagt entspricht eine Zeile Code einer Anweisung oder einem Befehl in der jeweiligen Programmiersprache.²³ Das Unix-Tool `wc` ermöglicht eine schnelle Zählung von Zeilen in Dateien. Der Aufruf `$ wc -l *.p *.inc *.a` gibt ein Total von 116.587 Zeilen für alle Pascal- und Assembler-Dateien an, also ungefähr die rund 128.000 Zeilen Code, die das CHM auf seiner Webseite verspricht. Aber `wc` liefert kein wirklich genaues Ergebnis. Denn Quelltexte enthalten oftmals auch leere Zeilen und Kommentare, um den Code zu strukturieren und zu erläutern. Lesen wir noch einmal die Datei `URotate.a`, diesmal Zeilen 10–28 mit Leerzeilen:

²³ Verschiedene Sprachen, aber auch individuelle Programmierstile verkomplizieren die Sache jedoch schnell.

```
10 ; *****
*****
11
12             SEG             'ADoRotate'
13
14 DoReverseBytes PROC             EXPORT
15
16 ;             Calling sequence (Pascal conventions):
17 ;
18 ;             PROCEDURE DoReverseBytes (data: Ptr;
19 ;                                     count: INTEGER);
20 ;
21 ;             Parameter Offsets
22
23 @data             EQU             10
24 @count            EQU             8
25
26 ;             Size of parameters
27
28 @params           EQU             6
```

Von den 19 hier wiedergegebenen Zeilen sind streng genommen nur gerade fünf als LOC oder *lines of code* zu zählen (Zeilen 12, 14, 23, 24 und 28), während die restlichen 14 Zeilen für menschliche Leser:innen des Codes gedacht sind und bei der Übersetzung des Programms in ausführbaren Maschinencode ignoriert werden.

Kommentare und leere Zeilen bei der Zählung zu ignorieren, ist mit `wc` allein nicht möglich. Dafür werden spezielle Tools benötigt. Eine gute Option ist das kostenlose Open-Source-Programm `cloc`.²⁴ Allerdings müssen wir es ein wenig anpassen. Weil `cloc` in den vorliegenden Assembler-Dateien

²⁴ Siehe <https://github.com/AlDanial/cloc> [Letzter Zugriff: 16.8.2023].

@-Zeichen fälschlich als Kommentarmarken interpretiert, korrigieren wir die Grundeinstellungen in einer eigenen Konfigurationsdatei (`--force-lang-def=myconfig`).²⁵ Außerdem müssen wir Dateien mit der Endung `.a` als Assemblersprache ausweisen (`--force-lang=Assembly,a`) und solche mit der Endung `.inc` als Pascal (`--force-lang=Pascal,inc`). Der etwas umständliche Befehl `$ cloc --force-lang-def=myconfig --force-lang=Assembly,a --force-lang=Pascal,inc *.a *.inc *.p` gibt uns dann eine detaillierte quantitative Analyse des Quelltextes: Pascal und Assembler-Sprache machen zusammen 36.937 leere Zeilen aus, 6.499 Zeilen mit Kommentaren und 73.151 Zeilen eigentlichen Code. Zählt man alles zusammen, kommt man exakt auf die 116.587 Zeilen, die uns schon `wc` genannt hatte.

Aber dank `cloc` wissen wir jetzt mehr. Fast die Hälfte aller Zeilen in Assemblersprache, nämlich 8.599 von insgesamt 18.746, sind entweder leer oder enthalten Kommentare; und auch ein Drittel aller Zeilen in Pascal, genau 32.070 von 97.841, sind leer, dienen also allein der visuellen Gliederung des Codes. `cloc` bestätigt zudem unseren ersten Eindruck, dass der Assembler-Code deutlich stärker kommentiert ist als die in Pascal geschriebenen Teile. Liegt das Verhältnis von Kommentar zu Code für Assemblersprache bei ungefähr 1:3,²⁶ so für Pascal gerade einmal bei rund 1:20.²⁷ Und das Verhältnis von Pascal zu Assemblersprache stellt sich, zählt man allein eigentliche LOC, mit 6:1 noch etwas ungleicher dar, als anfänglich auf Grundlage der Datei-

²⁵ Wir löschen aus der mit `cloc --write-lang-def=myconfig` erzeugten Datei die Anweisungen `filter remove_matches ^\s*\@` und `filter remove_inline \@.*$` im Abschnitt `Assembly` (Zeilen 120 und 125 mit Version 1.96 des Programms).

²⁶ Diese Aussage erfordert eine Einschränkung. Wie die oben angeführten Ausschnitte aus `URotate.a` zeigen, sind mit Semikolon auskommentierte Zeilen im Assembler-Code von Photoshop häufig leer oder mit Reihen von Sternchen gefüllt, erfüllen also keine erläuternde, sondern bloß strukturierende Funktion.

²⁷ Auch hier ist eine Ergänzung nötig: Tatsächlich enthalten deutlich mehr als die von `cloc` angegebenen Zeilen Kommentare, weil das Tool nur reine Kommentarzeilen zählt, am Ende einer Programmzeile eingefügte Kommentare aber ignoriert.

größen berechnet. Lässt man die Dateien vom Typ `.make`, `.r` und `.t` weg und vergleicht einzig LOC in Pascal und Assemblersprache, dann besteht Photoshop 1.0.1 zu 86 % aus Pascal.

Bevor wir uns im nächsten Schritt ganz den Inhalten der Dateien zuwenden, eine letzte Bemerkung zur Größe von Photoshop 1.0.1. Zählt man alle Zeilen aller Dateien, die den Quelltext ausmachen, inklusive Leerzeilen und Kommentare zusammen, kommt man auf ein Total von 128.602 Zeilen (`$ wc -l *.a *.inc *.make *.p *.r *.t`). Gemessen etwa an den 30 Millionen LOC des Chrome-Webrowsers erscheint das winzig.²⁸ Allerdings ist die Größe von Softwareprojekten seit den frühen 1990er-Jahren geradezu explodiert; und im Unterschied zu den Entwickler:innen-Teams heutiger Tage mit Dutzenden oder Hunderten Programmierer:innen wurde die erste Version von Photoshop im Wesentlichen von einer einzigen Person, Thomas Knoll, geschrieben.²⁹ Eine bessere Vergleichsgröße stellt deshalb vielleicht eine Anwendung wie MacPaint, der bahnbrechende Grafikeditor für den Apple Macintosh, dar, der 1984 ebenfalls von einer einzigen Person, Bill Atkinson, programmiert wurde. Nach Auskunft des CHM, das den Quelltext auch dieses Programms veröffentlicht hat, umfasst MacPaint insgesamt 9.405 LOC.³⁰ Photoshop war für die Zeit, rund fünf Jahre später, also wohl ein eher großes Projekt.

4.

Nun aber zu den Inhalten der Dateien – und nochmals die Frage: Wie können wir hunderttausend Zeilen Pascal- und

²⁸ Siehe https://openhub.net/p/chrome/analyses/latest/languages_summary [Letzter Zugriff: 15.8.2023].

²⁹ Siehe Shustek, „Adobe Photoshop Source Code“.

³⁰ Siehe Len Shustek, „MacPaint and QuickDraw Source Code“, Computer History Museum, Juli 2010, <https://computerhistory.org/blog/macpaint-and-quickdraw-source-code/> [Letzter Zugriff: 17.8.2023].

Assembler-Code lesen? Noch immer sollen nicht einzelne Stellen aus dem Programm herausgegriffen, sondern der Quelltext im Sinne des *distant reading* als bloße Masse von Wörtern oder Zeichen erfasst werden. Wir versuchen also, den Code mit maschinellen Mitteln so zu transformieren und zu reduzieren, dass dabei Muster, Beziehungen, Häufigkeiten und andere lexikalische oder textuelle Merkmale sichtbar werden. Im Folgenden ignorieren wir die Dateien vom Typ `.make`, `.r`, `.t` und `.txt` und beschäftigen uns ausschließlich mit den Pascal- und Assembler-Teilen von Photoshop.

Als Erstes trennen wir Code von Kommentar. Schließlich haben wir es im Quelltext mit zwei verschiedenen Ebenen zu tun, von denen eine Maschinen adressiert, die andere Menschen. Wir wollen die Kommentare von Thomas Knoll nicht mühsam aus Zehntausenden von Zeilen herausuchen, die auf fast zweihundert Dateien verstreut sind. Also führen wir alle Kommentare von den verschiedenen Orten zusammen – und zwar so, dass wir ihren Kontext im Quelltext rekonstruieren können. Für Assembler ist das im vorliegenden Fall leicht zu bewerkstelligen. Da Kommentare stets in einer Zeile stehen, die mit einem Semikolon anfängt, genügt eine einfache Mustererkennung mit dem Such- und Filtertool `grep`. Der Aufruf `$ grep '^;' -n *.a` liefert mit dem regulären Ausdruck `^;` alle gewünschten Zeilen samt ihrer Nummern und Dateinamen, beginnend mit:

```
UAbout.a:1;;Photoshop version 1.0.1, file: UAbout.a
UAbout.a:2;; Computer History Museum, www.computerhistory.org
UAbout.a:3;; This material is (C)Copyright 1990 Adobe Systems Inc.
UAbout.a:4;; It may not be distributed to third parties.
UAbout.a:5;; It is licensed for non-commercial use according to
UAbout.a:6;; www.computerhistory.org/softwarelicense/photoshop/
UAbout.a:11;; *****
*****
```

```
UAbout.a:17::          Calling sequence (Pascal conventions):
UAbout.a:18::
UAbout.a:19::          FUNCTION CodedChecksum (srcPtr: Ptr;
```

Um sicherzugehen, dass so alle Kommentarzeilen richtig erkannt werden, können wir das Resultat zählen: `$ grep '^;' *.a|wc -l` meldet 3.732 Zeilen, wie zuvor schon `cloc`.

Die Pascal-Dateien erfordern leider mehr Aufwand, da die Syntax für Kommentare komplex ist und Ausnahmen kennt. Kommentare stehen hier zwischen geschweiften Klammern, die oft über mehrere Zeilen verteilt sind. Solche Klammern können aber auch Teil von Anweisungen an den Compiler (in Kombination mit dem `@`-Zeichen) oder von druckbaren Zeichenketten sein, in beiden Fällen also zum eigentlichen Code gehören. Wir müssen daher ein kleines Programm schreiben, ein Shell-Skript, um Kommentar-Klammern von Code-Klammern zu unterscheiden:³¹

```
01 #!/bin/bash
02 # extracts_comments_p.sh
03 for file in $*; do
04   linum=0
05   while read -r line; do
06     ((linum=linum+1))
07     if [[ $line =~ { ]] \
08       && ! [[ $line =~ {\$ ]] \
09       && ! [[ $line =~ \'.*\' ]]
10       then comment="true"; fi
11     if [[ $comment == "true" ]]
```

³¹ Wie dieses Problem zeigt, kann der Einsatz von Open-Source-Programmen große Vorteile bringen. Während der Arbeit an dem hier diskutierten Skript fragte ich den Autor von `cloc` wegen einer entsprechenden Funktion für sein Tool an. Bereits nach wenigen Tagen hatte er sie in Version 1.99 von `cloc` ergänzt. Siehe <https://github.com/AIDanial/cloc/issues/750> [Letzter Zugriff: 21.8.2023].

```

12     then echo "$file":"$linum":"$line"; fi
13     if [[ $line =~ } ]]
14     then comment="false"; fi
15     done < "$file"
16 done

```

Das Skript geht alle gewünschten Dateien (Zeile 3) Zeile für Zeile durch (Zeile 5), zählt dabei mit (Zeile 6) und prüft, ob sich in der jeweiligen Zeile eine öffnende Klammer findet (Zeile 7), die *nicht* von einem Dollarzeichen gefolgt wird oder mit anderen Zeichen zwischen einfachen Anführungszeichen steht (Zeilen 8 u. 9);³² ist das der Fall, merkt sich das Skript, dass hier ein Kommentar beginnt (Zeile 10). Sieht das Skript im nächsten Schritt, dass es bei einer Kommentarzeile angekommen ist (Zeile 11), gibt es diese Zeile samt Namen der Datei und Nummer aus (Zeile 12). Bevor das Skript mit der nächsten Zeile oder Datei fortfährt, prüft es, ob in der aktuellen Zeile ein etwaiger Kommentar durch schließende Klammer beendet wird; wenn ja (Zeile 13), merkt sich das Skript das (Zeile 14).

Wir testen das Skript mit `$./extract_comments_p.sh *.inc *.p` und erhalten eine lange Ausgabe, die nach dem alphabetischen Durchgang durch alle Dateien wie folgt mit der Verarbeitung von `UVMemory.p` endet:

```

UVMemory.p:34:fLogicalSize: INTEGER;    { Logical size of each block }
UVMemory.p:35:fPhysicalSize: INTEGER;   { Physical size of each block }
UVMemory.p:37:fBlocksPerPage: INTEGER;  { Number of blocks per page }
UVMemory.p:38:fPageCount: INTEGER;      { Number of pages in array }
UVMemory.p:40:fPageList: HVMPagedList;  { List of pages holding array }
UVMemory.p:42:fData: Handle;            { Buffer to hold part of array }

```

³² Tatsächlich erfasst diese Logik nicht alle denkbaren Fälle korrekt, denn es könnte bspw. sein, dass in einer Zeile *sowohl* eine Klammer als druckbares Zeichen *wie auch* eine Kommentar-Klammer stehen. Solche Sonderfälle zu berücksichtigen, würde allerdings ein deutlich komplexeres Programm erfordern.

```
UVMemory.p:44:fDirty: BOOLEAN;           { Has the data in the buffer changed? }
UVMemory.p:46:fLoPage: INTEGER;           { First page in data buffer }
UVMemory.p:47:fHiPage: INTEGER;           { Last page in data buffer }
UVMemory.p:49:fNeedDepth: INTEGER;        { Depth of nested NeedPtr calls }
```

Eine Zählung der so gefundenen Kommentarzeilen mit `$./extract_comments_p.sh *.inc *.p|wc -l` ergibt ein Total von 2.844, also etwas mehr als die von `cloc` gemeldeten 2.767. Jedenfalls scheint unser Skript keine Zeilen ‚verpasst‘ zu haben. Mit einem leicht veränderten Aufruf von `grep` und dem Skript können wir die Kommentare nun speichern: `$ grep '^;' -n *.a > comments_a.txt` und `$./extract_comments_p.sh *.inc *.p > comments_p.txt` schreiben die entsprechenden Zeilen in zwei eigene Dateien. Für die weitere Lektüre haben wir jetzt den gesamten Kommentar zum Programm, fein säuberlich nach Sprachen getrennt und mit Angabe der jeweiligen Dateien und Zeilennummern, vorliegen.

Was sich schon beim ersten Extrahieren von Kommentaren aus den Assembler-Dateien gezeigt hatte, wird in den neu erzeugten Sammlungen besonders deutlich: Das CHM hat dem Quelltext einen sechszeiligen Kommentarblock vorangestellt, der sich in allen Dateien identisch wiederholt. Zudem finden sich in den Kommentaren viele Hundert Zeilen, die nur aus einer Reihe von Sternchen bestehen oder (mit Ausnahme der Kommentarmarke selbst) leer sind. Löschen wir alle diese Zeilen, die dem Quelltext nachträglich von Dritten hinzugefügt wurden oder bloß visuell strukturierende Funktionen erfüllen,³³ bleiben von den Assembler-Dateien 3.023 Zeilen übrig, von den Pascal-Dateien gerade einmal 462. Schrittweise

³³ Dies lässt sich schnell erreichen, wenn man den Aufruf von `grep` oder des Skripts in geeigneter Weise mit dem Stream-Editor `sed` kombiniert: `$ sed '/Photoshop version 1.0.1, file: '//,+5d` entfernt die sechszeiligen Kommentarblöcke des CHM, `$ sed '/[*]\{70,\}'/d` die Sternchenreihen, `$ sed '/[0-9];*[[space:]]*$/d` die leeren Kommentarzeilen.

haben wir über hunderttausend Zeilen Code auf rund 3 % der ursprünglichen Größe reduziert. Der Kommentar ist lesbar geworden.

Zuletzt können wir die Kommentare aus den Quelltext-Dateien löschen. Das geht glücklicherweise leicht, weil `cloc` dafür die Option `--strip-comments` bietet. Mit `$ cloc --strip-comments=nc --force-lang-def=myconfig --force-lang=Assembly,a --force-lang=Pascal,inc *.a *.inc *.p` legen wir von allen Dateien Kopien mit dem zusätzlichen Suffix `.nc` (*n[fo]c[omments]*) an, die nur noch ‚reinen‘ Code enthalten.

5.

Der durch `cloc` von Kommentaren bereinigte Quelltext umfasst, wie wir wissen, 73.151 Zeilen, davon über sechzigtausend in Pascal und gut zehntausend in Assemblersprache. Was machen wir damit?

Beschränken wir uns hier auf die Teile in Pascal. Da es sich dabei um eine höhere Programmiersprache handelt, sind sie einfacher zu verstehen als der Assemblercode. Aber sechzigtausend Zeilen sind noch immer viel zu viel, als dass wir einfach drauflos lesen könnten. Wie und wo fangen wir mit der Lektüre an? Eine Möglichkeit wäre, den Text mit Handbüchern und Webseiten zu Pascal und dem Apple Macintosh zu erschließen. Wir würden dann schnell herausfinden, dass wir es hier nicht mit gewöhnlichem Pascal zu tun haben, sondern mit einem objektorientierten Dialekt der Sprache, der bei Apple speziell für den Macintosh entwickelt wurde.³⁴ Wir würden verstehen, dass z.B. der Großbuchstabe U, mit dem die meisten Dateinamen der Pascal-Teile beginnen, für *unit* steht,

³⁴ Siehe Kurt J. Schmucker, *Object-Oriented Programming for the Macintosh*, Hasbrouck Heights, N. J.: Hayden Book Company 1986.

eine modulare Einheit von Pascal-Programmen. Wir würden lernen, Prozeduren und Funktionen, Typen und Instanzen von Objekten im Code zu erkennen. Wir könnten versuchen, mit diesem Wissen die gut zweitausend Zeilen der *unit URotate.p* durcharbeiten, um die darin definierten Objekte wie `TRotateFloatCommand` mit ihren Datenfeldern wie `fAngle` und Prozeduren wie `.IRotateFloatCommand` zu begreifen (Zeilen 71–75). Und wir könnten uns schließlich daranmachen, diesen Vorgang mit all den übrigen Pascal-Teilen des Programms, die uns interessant und wichtig scheinen, zu wiederholen. Wir wären Monate damit beschäftigt.

Eine andere Möglichkeit ist es, den Code nicht an ausgewählten Textstellen Zeile für Zeile zu lesen, sondern ihn in all seinen Teilen gleichzeitig und gleich, d.h. ungeachtet deren Stellung und Funktion im Quelltext, zu erfassen. Eine solche ‚oberflächliche‘ Lektüre hat den Vorteil, dass wir weiterhin naiv vorgehen können, ohne ein tieferes Verständnis des Programms und seiner Architektur oder einzelner Codeteile mitzubringen. Der Quelltext von Programmen besteht, wie jeder andere Text auch, aus ‚Wörtern‘. Es sind das erstens die reservierten Schlüsselwörter der jeweiligen Sprache,³⁵ also ihre fest eingebauten Befehle, Zeichen für logische und arithmetische Operationen und andere syntaktische Marken wie Klammern, Semikola usw. Zweitens, und von besonderer Bedeutung, sind es die von den Programmier:innen gewählten (und hoffentlich möglichst deskriptiven) Namen für Variablen, selbst geschriebene Funktionen usw. – etwa das oben erwähnte Objekt `TRotateFloatCommand`. Drittens kommen die, in der Regel normalsprachlichen, Wörter aus druckbaren Zeichenketten für die Ausgabe hinzu. Schließlich bilden, wenig erstaunlich, Zahlen zur Berechnung verschiedener Größen einen wichtigen Bestandteil von Code. Behandeln wir Quelltext nun nicht als wohlge-

³⁵ Einige besondere Sprachen wie z.B. Prolog kennen keine Schlüsselwörter.

ordneten Zusammenhang solcher Wörter (eben als Text), sondern als zunächst ungeordnete Zusammenstellung oder Liste einzelner Elemente, dann können wir anfangen, Aussagen über den ganzen Code zu machen, ohne ihn ganz lesen zu müssen. Welches sind die Wörter, aus denen Photoshop 1.0.1 besteht? Wie viele sind es insgesamt? Wie viele unterschiedliche Wörter gibt es? Welche davon kommen häufiger vor? Welche tauchen dagegen nur selten auf? Welche sind besonders lang, welche besonders kurz?

Um solche Fragen zu beantworten, formen wir die von Kommentaren bereinigten Pascal-Dateien als Erstes zu einem großen Datenstrom um, in dem jedes Wort auf einer eigenen Zeile steht. Dazu geben wir den Inhalt aller Dateien nacheinander aus (`$ cat *.inc.nc *.p.nc`) und ersetzen dabei fortlaufend alle worttrennenden Zeichen, d.h. Leer- und Tabulatorzeichen durch Zeilenumbrüche (`|tr -s "[\t]" '\n'`). Der resultierende Datenstrom endet mit folgenden Zeilen:

```
INTEGER);
TVMArray;
PROCEDURE
VMCompress
(complete:
BOOLEAN);
IMPLEMENTATION
{$I
UVMemory.inc1.p}
END.
```

Unschwer erkennen wir darin die Wörter aus den letzten Zeilen der von `grep` zuletzt verarbeiteten Datei `$ tail UVMemory.p.nc`:


```
64 FUNCTION NewLargeHandle (size: LONGINT): Handle;
65 PROCEDURE ResizeLargeHandle (h: Handle; size: LONGINT);
66 PROCEDURE FreeLargeHandle (h: Handle);
67 FUNCTION NewVMArray (count: LONGINT;
68                     size: INTEGER;
69                     interleave: INTEGER): TVMArray;
70 PROCEDURE VMCompress (complete: BOOLEAN);
71 IMPLEMENTATION
72 {$I UVMemory.inc1.p}
73 END.
```

Wie viele Wörter haben wir so gefunden? Die Ergänzung unseres angepassten Befehls um `|wc -l` verrät es: 193.835. Die Zahl passt nicht schlecht zu den vorhin festgestellten gut 70.000 LOC: Auf jeder Zeile Quelltext stehen im Schnitt zwei bis drei Wörter.

Ein zweiter Blick auf das Ende unseres Datenstroms und der Datei `UVMemory.p.nc` zeigt, dass wir die Wortliste aber in zweierlei Hinsicht nachbearbeiten müssen. Erstens sollten die Wörter vom Beiwerk der Sonderzeichen wie Klammern oder Doppelpunkte befreit werden; und zweitens wollen wir Wiederholungen von Wörtern berücksichtigen (siehe etwa die wiederkehrenden Ausdrücke `FUNCTION`, `size`, `Handle` usw. im Ausschnitt aus `UVMemory.p.nc` oben). Letzteres ist leicht, wenn Ersteres geschafft ist. Das aber ist schwierig. Die Syntax von Pascal ist komplex, und man muss viele Ausnahmen berücksichtigen, damit bei der Zeichenverarbeitung nichts schief läuft (z.B. negative Vorzeichen und Dezimalpunkte von Zahlen oder Erweiterungen in Dateinamen wie `UVMemory.inc1.p` falsch interpretiert werden).

Macht man sich die Mühe, den Datenstrom mit Tools wie `sed`, `sort` und `uniq` so aufzubereiten, dass möglichst keine Information verloren geht, dann kommt man nach einigem Probieren auf ein Skript wie das folgende, das mit seinen regu-


```
894  2 ?
895  2 ????
896 531 @
897 110 A
898  15 a
899  21 a1
900  21 a2
901  21 a3
902   5 a3by3Filter
903   1 Aacute
```

Zeichenketten wie ????, Operatoren wie @, kürzeste Ausdrücke wie A, a oder a1 (wahrscheinlich Teile druckbarer Zeichenketten sowie der Name einer Variable) und Wörter wie a3by3Filter (vielleicht der Name einer Prozedur) stehen zusammen mit der Anzahl ihres Vorkommens in zwei Spalten. Und mit dieser Liste können wir uns nun anschicken, die hunderttausend Zeilen Code (oder wenigstens die 63.004 Zeilen Pascal) von Photoshop 1.0.1 auf neue Art zu lesen.

Solche Lektüren setzen, das sollte klar geworden sein, immer mit Überlegungen quantitativer Art an. Beginnen wir mit der einfachsten aller Fragen: Aus wie vielen verschiedenen Wörtern samt Zahlen, Operatoren usw. setzt sich der Pascal-Code zusammen? `$ wc -l words_p.txt` sagt es uns: 7.930. Wir haben die ursprüngliche Zahl von 193.835 also auf weniger als 5 % der Menge reduziert. Was können uns diese knapp achttausend Wörter weiter verraten? Die Wortliste lässt sich mit `sort` auf verschiedene Art sortieren, mit `grep` können dabei bestimmte Zeichenketten aufgefunden oder ausgeschlossen und mit `awk` zudem Berechnungen angestellt werden. In Kombination erlauben es die Tools so, auch komplexere Fragen an den Quelltext zu stellen – zum Beispiel die naheliegende, wie viele ‚eigentliche‘ Wörter, also nicht Zahlen oder Operatoren, der Code enthält. Der Aufruf `$ awk '$2 ~ /^[[:alpha:]]/'`

words_p.txt|wc -l, der in der zweiten Spalte der Wortliste nach Einträgen sucht, die mit einem alphabetischen Zeichen beginnen, nennt uns ein Total von 7.018 verschiedenen Wörtern; der um eine kleine arithmetische Operation ergänzte Befehl `$ awk '$2 ~ /^[[:alpha:]]/ {SUM+=$1} END {print SUM}' words_p.txt` ergibt eine Summe von 163.236 Nennungen dieser Wörter in allen Pascal-Dateien des Quelltextes.

Weitere Fragen, die uns die Wortliste zu beantworten hilft, wären etwa:

- Welches sind die 30 häufigsten Wörter? (`$ grep '[[[:alnum:]]\+$' words_p.txt|sort -k1nr|head -n 30`)
- Welche Wörter, mit Ausnahme von Zahlen, werden dagegen nur ein einziges Mal im Code genannt? (`$ grep '\<[[[:alnum:]]\>$' words_p.txt|awk '$1 == 1'`)
- Was für Zahlen, binär oder dezimal notiert, sind im Code zu finden? (`$ grep '\-?\([[:0-9]]+\)\(\.[[:0-9]]+\)\{0,1\}$' words_p.txt`)
- Welches sind die zehn längsten Wörter? (`$ awk '{print length " " $0}' words_p.txt|sort -nr|head`)
- Welche Wörter sind großgeschrieben, welche kleingeschrieben? (`$ grep '\<[[[:upper:]][[[:lower:]]\+>' words_p.txt` und `$ grep '\<[[[:lower:]]\+>' words_p.txt`)
- Wie oft werden die reservierten Schlüsselwörter von Pascal insgesamt verwendet?³⁷ (`$ grep -wf reserved_p.txt words_p.txt|awk '{SUM+=$1} END {print SUM}'`)
- Welche nur in Großbuchstaben geschriebenen Wörter sind *nicht* reservierte Schlüsselwörter? (`$ grep -vwf reserved_p.txt words_p.txt|grep '\<[[[:upper:]]\+>'`)

³⁷ Dafür wird eine zusätzliche Datei reserved_p.txt mit allen Schlüsselwörtern benötigt, die man bspw. auf folgender Webseite aufgeführt findet: https://wiki.free-pascal.org/Reserved_words [Letzter Zugriff: 23.8.2023].

Natürlich ist damit allein noch lange keine Codekritik getan. Aber mit solchen und ähnlichen Fragen bekommt man schnell ein gutes Gespür für den Quelltext – und eben für den *gesamten* Quelltext –, ohne ihn schon verstehen oder mühsam Zeile für Zeile studieren zu müssen.

Ein Beispiel: Die fünf am häufigsten genannten Zahlen sind 0, 1, 2, 3 und 4 – und zwar genau in dieser Reihenfolge! Das ist vielleicht nicht allzu erstaunlich. Aber dass bald danach die 255 folgt, lässt immerhin vermuten, dass das Programm für viele interne Berechnungen nicht mehr als ein Byte in Anspruch nimmt. Dagegen sind nur ein einziges Mal auftauchende Zahlen wie 57,295779513 oder 2147483647 wenigstens auf den ersten Blick so ungewöhnlich, dass sie zu Nachforschungen einladen. Ähnlich bei alphabetischen Wörtern: Die fünf meist gebrauchten sind r, doc, view, row und h, wohl alles Namen von Variablen. Das drückt einerseits, wiederum wenig überraschend, das Bestreben des Programmierers aus, ständig sich wiederholende Elemente knapp zu notieren (und also weniger tippen zu müssen); andererseits sind r und h als Namen so nichtssagend, dass sie geradezu auffordern, den Quelltext genauer auf ihre Verwendung hin zu lesen – und dann zu erkennen, dass bspw. r fast immer eine Variable vom Typ Rect darstellt, eines rechteckigen Bildelements, das einen Grundbaustein des Grafiksystems damaliger Macintosh-Computer darstellt. Umgekehrt stehen unter den ‚einzigartigen‘ Wörtern im Code solche wie zsbanswzmgpsmdp natürlich ins Auge und werfen entsprechende Fragen auf.

6.

Schließlich können wir damit beginnen, gezielt einzelne Teile aus dem Quelltext herauszugreifen. Dabei mag entweder die zuvor erstellte Wortliste als Inspirationsquelle dienen, die uns

hoffentlich auf interessante Namen, Zahlen oder andere Zeichenketten gebracht hat. Oder wir können bestimmte Strukturen oder Elemente im Code in den Fokus nehmen, ohne schon deren einzelne Namen aus einer Wortliste o.Ä. zu kennen. Die prominentesten Beispiele dafür sind wohl Variablen, Prozeduren und Funktionen sowie – im vorliegenden Fall – sogenannte Objekte. Dieses Vorgehen erfordert jedoch, anders als das naive Zusammenstellen und Analysieren der Wortliste, ein Mindestmaß an Verständnis der jeweiligen Programmiersprache. Das soll zum Schluss an den erwähnten Objekten, besonderen informatischen Konstrukten, demonstriert werden. Wir wählen diesen Gegenstand erstens, weil Objekte das definierende Merkmal der Programmiersprache im Dialekt Object Pascal sind (der Name kommt nicht von ungefähr).³⁸ Die Kombination von Datensätzen und zugehörigen Methoden ihrer Verarbeitung ist typisch für das Design von Photoshop wie von moderner Anwendungssoftware überhaupt. Zweitens aber nehmen wir Objekte hier aus einem ganz pragmatischen Grund in den Blick: Sie sind im Quelltext sehr leicht auszumachen, weil die Syntax ihrer Deklaration, anders als etwa bei Variablen, einfach und eindeutig ist.

Jede Deklaration eines Objekts geschieht nach demselben schlichten Muster: Dem Schlüsselwort `TYPE` folgen die Namen der Objekte, jeweils mit dem Zusatz `= OBJECT`. Es genügt daher eine einfache Suche nach diesem Muster mit `$ grep '= OBJECT' *.p.nc`.³⁹ Schauen wir auf den Anfang der so erzeugten Liste (`|head`), dann sehen wir Folgendes:

³⁸ Siehe Ken Doyle, „Introduction to Object Pascal“, in: *MacTech: The Journal of Apple Technology* 2:12 (1985), <http://preserve.mactech.com/articles/mactech/Vol.02/02.12/ObjectPascal/index.html> [Letzter Zugriff: 25.8.2023]; Schmucker, *Object-Oriented Programming for the Macintosh*.

³⁹ Die `.inc`-Dateien können wir in diesem Fall ignorieren, da Objekte ausschließlich in `.p`-Dateien deklariert werden.

```
UAbout.p.nc: TSerialText = OBJECT (TKeyHandler)
UAbout.p.nc: TRegisterDialog = OBJECT (TBWDialog)
UAdjust.p.nc: TAdjustmentCommand = OBJECT (TFloatCommand)
UAdjust.p.nc: TInvertCommand = OBJECT (TAdjustmentCommand)
UAdjust.p.nc: TEqualizeCommand = OBJECT (TAdjustmentCommand)
UAdjust.p.nc: TFeedbackDialog = OBJECT (TBWDialog)
UAdjust.p.nc: THistDialog = OBJECT (TFeedbackDialog)
UAdjust.p.nc: TThresholdDialog = OBJECT (THistDialog)
UAdjust.p.nc: TThresholdCommand = OBJECT (TAdjustmentCommand)
UAdjust.p.nc: TPosterizeDialog = OBJECT (TFeedbackDialog)
```

Nun sind uns die hier ausgegebenen Namen der Objekte (TSerialText, TRegisterDialog, TAdjustmentCommand usw.) insofern bereits ‚bekannt‘ und nichts Neues, als sie natürlich auch in der zuvor erstellten Wortliste stehen. Aber diese Aufstellung, die den Kontext der jeweiligen Codezeile bewahrt, verrät uns doch mehr. Die Deklaration jedes Objekts mit dessen Namen nennt nämlich in Klammern einen weiteren Namen, der sich bei näherer Betrachtung als der eines *anderen* Objekts entpuppt. So verweist etwa das Objekt TThresholdDialog in der drittuntersten Zeile der Ausgabe auf das Objekt THistDialog, das selbst in der Zeile gleich darüber deklariert wird. Hier zeigt sich eine der wichtigsten Eigenschaften von Objekten: Sie können aus anderen Objekten abgeleitet werden (im Jargon der Informatik heißt das ‚Vererbung‘).

Die Objekte eines Programms stehen zueinander auf verschiedene Arten in Beziehung, wovon ihre Verwandtschaft durch Ableitung die wohl bedeutendste ist. Ein zweiter Blick auf die Ausgabe oben macht das klar. Das Objekt THistDialog, von welchem das Objekt TThresholdDialog abgeleitet wird, ist seinerseits – wie gesagt – vom Objekt TFeedbackDialog abgeleitet (viertunterste Zeile), und dieses wiederum (sechste Zeile) vom Objekt TBWDialog, wie das Objekt TRegisterDialog (zweite Zeile) auch. Wir haben es bei den Objekten im Code

von Photoshop also mit einer vielstufigen Hierarchie von Verwandtschaftsbeziehungen zu tun. Diese Hierarchie der Objekte macht ganz wesentlich die Struktur des Programms aus. Wollen wir den Aufbau des Quelltextes verstehen, müssen wir diese Struktur kennen. Sie ist aber alles andere als leicht zu überschauen. Eine schnelle Zählung (`|wc -l`) ergibt ein Total von 176 Objekten. Wir dürfen davon ausgehen, dass die am Beispiel der ersten Objekte in der Ausgabe sichtbar gewordene Verwandtschaft weiter reicht. Aber wie viel weiter und wohin? Und wie sieht sie im Einzelnen aus? Wie sollen wir die Hierarchie der Objekte rekonstruieren?

Wir könnten die Aufgabe händisch mit Stift und Papier angehen. Oder wir können das den Computer besorgen lassen. Dazu verändern wir die bereits erzeugte Liste so, dass sich in ihr nur noch die Namen der Objekte befinden, wobei der Name eines abgeleiteten Objekts in der Zeile immer links neben dem Namen seines Ursprungsobjekts steht. Wir wandeln unseren anfänglichen Aufruf von `grep` daher leicht ab zu `$ grep -h '= OBJECT' *.p.nc|tr -d '\t()' |awk -F' ' '{print $2 " " $4}'`. Damit werden die Dateinamen am Zeilenanfang unterdrückt, die Klammern gelöscht und die zwischen den verbleibenden Leer- und Gleichzeichen an zweiter und vierter Stelle stehenden Namen der Objekte herausgefiltert. Die Liste beginnt nun neu so:

```
TSerialText TKeyHandler
TRegisterDialog TBWDialog
TAdjustmentCommand TFloatCommand
TInvertCommand TAdjustmentCommand
TEqualizeCommand TAdjustmentCommand
TFeedbackDialog TBWDialog
THistDialog TFeedbackDialog
TThresholdDialog THistDialog
TThresholdCommand TAdjustmentCommand
TPosterizeDialog TfeedbackDialog
```


Mit dem Zusatz `> objects.txt` schreiben wir die Liste in eine eigene Datei. Als Nächstes schreiben wir ein kurzes Skript, das die Liste durcharbeitet und versucht, den am rechten Ende der Zeilen stehenden Ursprungsobjekten die Namen der wiederum ihnen entsprechenden Ursprungsobjekte aus anderen Zeilen zuzuordnen.

```
01 #!/bin/bash
02 # make_object_hierarchy.txt
03 while read -r line; do
04   parent=$(echo $line | awk '{ print $NF }')
05   grandparent=$(awk "/^$parent/ { print \$2 }" $1)
06   if [[ $grandparent != "" ]]
07     then echo -e "$line $grandparent"
08   else echo "$line"; fi
09 done < $1
```

Das Skript geht die Eingabe, also Datei, Zeile für Zeile durch (Zeile 3), speichert dabei den am Zeilenende rechts stehenden Namen als ‚Elternteil‘ (Zeile 4) und durchsucht dann die gesamte Liste darauf hin, ob der eben gefundene Name an einer anderen Stelle ganz links, d.h. am *Zeilenanfang* steht, wobei das Ergebnis nun als ‚Großelternteil‘ zwischengespeichert wird (Zeile 5); ist das Ergebnis positiv, der Name des ‚Großelternteils‘ also nicht leer (Zeile 6), dann wird die aktuell erfasste Zeile um diesen Namen rechts ergänzt (Zeile 7), ansonsten – wenn es also keinen Namen eines ‚Großelternteils‘ gibt – bleibt sie, wie sie ist (Zeile 8).

Wir rufen das Skript mit `$./make_object_hierachy.sh objects.txt` auf und erhalten als Ausgabe die um eine Stufe in den Verwandtschaftsbeziehungen der Objekte erweiterte Liste, die folgendermaßen beginnt:

```
TSerialText TKeyHandler  
TRegisterDialog TBWDialog TDialogView  
TAdjustmentCommand TFloatCommand TBufferCommand  
TInvertCommand TAdjustmentCommand TFloatCommand  
TEqualizeCommand TAdjustmentCommand TFloatCommand  
TFeedbackDialog TBWDialog TDialogView  
THistDialog TFeedbackDialog TBWDialog  
TThresholdDialog THistDialog TFeedbackDialog  
TThresholdCommand TAdjustmentCommand TFloatCommand  
TPosterizeDialog TFeedbackDialog TBWDialog
```

Wie wir sehen, hat das Skript für jedes Objekt außer für das in der ersten Zeile stehende `TSerialText` ein ‚Großelternteil‘ gefunden. Und es sind, wie etwa die zweite Zeile zum Objekt `TRegisterDialog` belegt, die richtigen. Schreibt man diese neue Liste wieder in eine Datei, z.B. mit dem Zusatz `> objects_1.txt`, kann man das Skript nochmals aufrufen. Jede weitere Ausführung, die als Eingabe die Ausgabe des vorigen Schritts verarbeitet, ergänzt die Liste um eine zusätzliche Ebene der Hierarchie – bis die Verwandtschaftsbeziehungen nach dem vierten Schritt oder der Ausgabe in die Datei `objects_4.txt` erschöpft, d.h. komplett erfasst sind (was beim ersten Objekt auf der Liste, wie gesehen, überhaupt keiner Ergänzung bedurfte).

Die jetzt in jeder Zeile ganz rechts stehenden Objekte werden im Quelltext also nur aufgerufen, nicht selbst deklariert. Sie haben ihre Herkunft nicht in Photoshop 1.0.1, sondern stammen, wie eine Internetrecherche zeigt, aus einer Softwarebibliothek namens `MacApp`, dem seinerzeit offiziellen Framework von Apple für die Entwicklung von Macintosh-Programmen mit Object Pascal.⁴⁰ Die Objekte dieser Bibliothek

⁴⁰ Siehe basalgangster, „Building Photoshop“, März 2013, http://basalgangster.macgui.com/RetroMacComputing/The_Long_View/Entries/2013/3/30_Building_Photoshop.html [Letzter Zugriff: 24.8.2023]; <https://en.wikipedia.org/wiki/MacApp> [Letzter Zugriff: 24.8.2023].

bilden gewissermaßen das informatische Fundament, auf dem Anwendungssoftware, die Apples Vorgaben folgte, ruht. Welche sind das? Der Befehl `$ grep -o '\<[[:alpha:]]\+$' objects_4.txt|sort|uniq -c`, der jeweils das letzte Wort am Ende einer Zeile findet, gibt sie uns gezählt aus:

```
1 TApplication
109 TCommand
4 TDialogItem
20 TDialogView
1 TDocument
3 TFrame
3 TKeyHandler
23 TObject
1 TPrintStyleChangeCommand
1 TStdPrintHandler
8 TView
2 TWindow
```

Die meisten Namen sind sprechend: Anwendung, Befehl, Dialog, Dokument, Fenster. Nach diesen zwölf Grundbausteinen sind alle Objekte von Photoshop modelliert und ist das Programm damit gewissermaßen im Baukastenprinzip zusammengefügt. Manche MacApp-Objekte, wie `TApplication`, finden dabei nur ein einziges Mal Verwendung; andere dienen wiederholt als Ausgangspunkt für die Konstruktion eigener Programmkomponenten. Mit über 100 abgeleiteten Objekten für Befehle bildet `TCommand` so etwas wie den informatischen Fluchtpunkt der Funktionalität von Photoshop.

Mit 176 Einträgen ist die Liste der aus MacApp abgeleiteten Objekte in Photoshop samt all ihrer ‚Eltern‘, ‚Großeltern‘, ‚Urgroßeltern‘ usw. bis zu den zwölf ‚Urformen‘ zurück nicht annähernd so umfangreich wie die Wortliste `words_p.txt`. Und entsprechend geringer sind die Möglichkeiten ihrer Lektüre.

Aber auch sie bietet einige Einsichten in die Architektur des Programms, die mögliche Ansatzpunkte für weitere Analysen liefern. Fragen, die man an sie stellen kann, sind etwa:

- Wie viele ‚Elternteile‘ haben die einzelnen Objekte? (`$ awk '{print NF " " $0}' objects_4.txt|sort -k1nr`)
- Was sind die rudimentären, d.h. in nur einem Schritt aus MacApp-Objekten abgeleiteten Befehle, die Photoshop implementiert? (`$ awk 'NF <= 2 && /TCommand/ {print $1}' objects_4.txt`)
- In welchen ‚Ansichten‘ und ‚Rahmen‘ stellt sich Photoshop auf der grafischen Benutzeroberfläche dar? (`$ grep '\(TView\|TFrame\)' objects_4.txt`)
- Mit etwas Programmiergeschick ließe sich die Liste zudem als Baumdiagramm oder auf andere anschauliche Weisen ausgeben.

7.

Unser *distant reading* des Quelltextes von Photoshop I.O.I hat über hunderttausend Zeilen Code auf gut siebzigtausend Zeilen Anweisungen, rund dreieinhalbtausend Zeilen Kommentar, siebentausend verschiedene Pascal-Wörter, knapp zweihundert Objekte und zuletzt auf zwölf elementare Typen reduziert.

Wir haben an der ‚Oberfläche‘ des Quelltextes mit dem Offensichtlichen begonnen (den Dateinamen und -größen), zunächst einfache Kommandozeilen-Werkzeuge eingesetzt (wie `ls` zum Zählen der Zeilen), dann mit mächtigeren Tools gearbeitet (wie `sort`, `grep` und `sed` für die Sortierung, Muster-suche und Transformation von Zeichenketten) und schließlich eigene kurze Skripte für gefilterte Listen erstellt. Der Code wurde so auf verschiedenen Ebenen (von Dateien, Zeilen, Anweisungen und Kommentaren, einzelnen Wörtern und

bestimmten Objekten) maschinell gelesen, prozessiert und umgeschrieben.

Dabei sind einerseits besondere Bestandteile des Quelltextes sichtbar geworden, die zu näherer Betrachtung einladen (von ganzen Dateien wie `Photoshop.r` über algorithmische Verfahren wie Huffman-Codierung und basale Objekte wie `TCommand` bis hin zu einmalig auftauchenden Zeichenketten wie `zsBanS-wzaMGpSmDP` oder Zahlen wie `57,295779513`). Andererseits hat unsere Lektüre mehrere nicht bloß technische, sondern auch soziokulturell geprägte Aspekte des Programms beleuchtet, deren nachfolgende Analysen und Interpretationen wir dank des auf verschiedene Arten aufbereiteten und reduzierten Codes nun leichter auf bestimmte Strukturen und Elemente im Quelltext beziehen können. Um hier nur einige der wichtigsten zu nennen: die Implementierung von Datenstrukturen und Prozeduren in unterschiedlichen Programmiersprachen und -paradigmen (wie Object Pascal und Assembler bzw. strukturierter und maschinennaher Programmierung); die Frage der Autor:innenschaft am Code (von einzelnen Programmierer:innen wie Thomas Knoll zu den Entwickler:innenteams heutiger Tage); die Stellung von Kommentaren in Quelltexten; die Namensgebung von Variablen, Prozeduren oder Funktionen und Objekten; die zentrale Rolle der Objektorientierung für moderne Anwendungsprogramme; oder die Bedeutung von Frameworks für die Softwareentwicklung (wie MacApp).

Wie bereits mehrfach gesagt, kann ein *distant reading* großer Quelltexte, wie es hier beispielhaft vorgeführt wurde, deren Kritik nur vorbereiten oder ihr zuarbeiten. Wie aussagekräftig eine solche Kritik im Falle von Anwendungsprogrammen wie Photoshop angesichts der scheinbar puren Funktionalität des Codes überhaupt sein kann, müssen weitergehende Analyse erst zeigen. Dass man hunderttausend Zeilen Code aber durchaus lesen kann, ist hoffentlich deutlich geworden.

PROGRAMMIEREN MIT COPILOT. ÜBER GRENZEN DER AUTOMATISIERUNG UND FORMEN VON SUBJEKTIVIERUNG

Dan Verständig

1. Einleitung

Technologische Entwicklungen rund um Künstliche Intelligenz (KI) haben seit einigen Jahren eine gesteigerte Aufmerksamkeit in gesellschaftlichen, politischen, aber auch wissenschaftlichen Diskursfeldern bekommen.¹ Dabei ist KI an sich keine neue Erscheinung. Es sind eher die Rahmenbedingungen und gesteigerten Rechenleistungen sowie die aktuellen Möglichkeiten rechenbasierte Architekturen zu realisieren, die diesem Thema jüngst zum Durchbruch verholfen haben. Schon länger wird KI eingesetzt, um alltägliche Abläufe und Prozesse zu unterstützen, beispielsweise beim Schreiben von E-Mails, bei der Sortierung von Fotoalben oder bei der automatisierten Bestellung von Produkten. Bislang hat KI jedoch nicht direkt zur Verbesserung von Programmcode beigetragen, sodass der Prozess der Softwareentwicklung fast vollständig manuell erfolgte. Das ändert sich gerade, denn KI findet sich inzwischen auch in integrierten Entwicklungsumgebungen (IDE) wie Visual Studio Code und JetBrains IntelliJ wieder, um einerseits Routineaufgaben zu automatisieren und andererseits den geschriebenen Code zu verbessern. Dabei wird auf große Sprachmodelle wie GPT-4 oder beispielsweise auch Codex von OpenAI und AlphaCode von Deepmind gesetzt. Diese können

¹ Selena Nemorin u.a., „AI Hyped? A Horizon Scan of Discourse on Artificial Intelligence in Education (AIED) and Development“, in: *Learning, Media and Technology* 48:1 (2023), S. 38–51.

Programmcode erzeugen und damit Probleme lösen, die in natürlicher Sprache formuliert sind. Diese Technologie wurde bereits in mindestens einer weitverbreiteten Programmiereditor-Erweiterung erfolgreich kommerzialisiert: dem GitHub Copilot. Daneben stellen Tools wie Tabnine Kite und der Amazon Code Whisperer weitere Ansätze dar, um die Produktivität von Entwickler:innen zu erhöhen. Trotz der verstärkten Hinwendung zu KI-basierten Systemen in der Softwareentwicklung ist relativ wenig über die Implikationen bekannt, wie sich die Programmierung und die Kommunikation verändern, welche Konsequenzen der Einsatz von KI kurz-, mittel-, und langfristig für die Entwicklung hat und wie sich die Technologie auf das Erlernen von Programmierparadigmen und -sprachen sowie Problemlösungen auswirkt.

Neil A. Ernst und Gabriele Bavota skizzieren hier einige Aspekte, die insbesondere im Hinblick auf Praktiken des Programmierens wichtig sind.² Sie nennen Biases (sowohl bei den Trainingsdaten als auch bei den Menschen in der Nutzung), gesetzliche Rahmenbedingungen (urheberrechtliche und lizenzrechtliche Fragen) und Sicherheitslücken. Zudem heben sie auch hervor, dass es zu berücksichtigen gilt, wie sich die Art und Weise des Lernens verändern kann. Advait Sarkar u.a. haben aufgezeigt, dass der Einsatz von großen Sprachmodellen zur Unterstützung bei der Programmierung die Art und Weise grundlegend verändert, wie Menschen Programmcode schreiben.³ Der Einsatz von KI verändere, so die Autor:innen des Microsoft Research Departments, fast jeden Aspekt der Programmiererfahrung, einschließlich Planung, Autor:innenschaft, Wiederverwendung, Änderung, Verständnis und Fehlersuche. Sie weisen auch darauf hin, dass die Unterstützung von großen

² Neil A. Ernst und Gabriele Bavota, „AI-Driven Development Is Here: Should You Worry?“, in: *IEEE Software* 39:2 (2022), S. 106–110.

³ Advait Sarkar u.a., „What Is It Like to Program with Artificial Intelligence?“, in: *arXiv*, August 2022, DOI: 10.48550/arXiv.2208.06213.

Sprachmodellen bei der Programmierung eine „ganz besondere Note“⁴ habe und sich damit neue Herausforderungen für die Forschung darüber ergeben, wie Menschen am Code und über den Code zusammenarbeiten. Das betrifft einerseits die Praktiken des Schreibens von Code, aber auch die Fragen der kollaborativen Lösung von Problemen. Sie weisen zudem darauf hin, dass es eine noch größere Herausforderung sei, Menschen zu helfen, die Technologien zu verstehen und Potenziale auf Anwendungsebene zu entfalten, die keine Expert:innen für den Einsatz von KI sind.⁵ Wie sich die hier beschriebene besondere Qualität auszeichnet, soll in diesem Beitrag genauer diskutiert werden.

Die Entwicklungen rund um KI und Programmierung sind vor dem Hintergrund der gesellschaftlichen Strukturveränderungen auch aus erziehungswissenschaftlicher Perspektive relevant, denn die Förderung von digitalen Kompetenzen und *coding literacy*⁶ sind in den vergangenen Jahren vielerorts wieder laut geworden. Das ist angesichts der raschen Verbreitung des Computers und vernetzter Architekturen sowie der damit verbundenen Durchdringung in nahezu alle gesellschaftlichen Bereiche wenig verwunderlich. Wer programmieren kann, der kann die Welt neu schreiben. Die Gestaltung von digitalen Architekturen und die Entwicklung von ganzen Infrastrukturen der Vernetzung und Kommunikation sind heute dynamische und attraktive Felder, in denen sich Menschen verwirklichen können, und es sind nicht selten Bereiche, die sich durch innovative Ansätze in der Wissensarbeit und Wissensvermittlung auszeichnen. Darüber hinaus ist das Erlernen von Programmierkenntnissen inzwischen auch ein bildungspolitisches Thema. Es hat gewissermaßen eine Renaissance erfah-

⁴ Ebd., S. 19.

⁵ Ebd.

⁶ Annette Vee, *Coding Literacy: How Computer Programming Is Changing Writing*, Cambridge, MA: The MIT Press 2017.

ren.⁷ Für die Bildungspolitik muss der digitale Wandel so gestaltet werden, dass er Menschen ermächtigt, sich in der digitalen Welt zu orientieren. Programmierkenntnisse und ein grundlegendes Verständnis über digitalen Architekturen und ihre Auswirkungen auf das Soziale sind daher heute besonders relevant. Diese und ähnliche digitale Kompetenzen tauchen nach wie vor in unterschiedlichen Kompetenzrastern auf, unabhängig davon, inwiefern sich die Programmiererfahrung durch den Einsatz von KI verändert. Das hat sowohl auf nationaler als auch internationaler Ebene Konsequenzen für die Gestaltung von Lern- und Bildungssettings, die sowohl direkt als auch indirekt mit der Vermittlung von Programmierkenntnissen in Verbindung stehen. Während unterschiedliche Positionspapiere nahezu unermüdlich und inzwischen seit mehr als 25 Jahren auf die Notwendigkeit hinweisen, Medienkompetenz und jüngst immer häufiger auch digitale Kompetenzen zu fördern,⁸ scheinen Erfolge in der Umsetzung von Maßnahmen eher ernüchternd. Obwohl vielfach Förderprogramme vor dem Hintergrund der Digitalisierung angestoßen wurden, zeichnet die evidenzbasierte Bildungsforschung inzwischen eine dilemmatische Situation nach, bei der die Kompetenzniveaus insbesondere in der schulischen Bildung insgesamt sinken.

Die Problematik ist komplex, sie lässt sich zum einen an der Trägheit von Bildungsinstitutionen beschreiben, manifestiert sich zum anderen jedoch auch in einem reduktionistischen

⁷ Shahira Popat und Louise Starkey, „Learning to Code or Coding to Learn? A Systematic Review“, in: *Computers & Education* 128 (2019), S. 365–376.

⁸ Es lassen sich hier verschiedene Positionen und Entwicklungen beschreiben, die kursorisch an den folgenden Papieren herausgestellt werden können: KMK [Sekretariat der Ständigen Konferenz der Kultusminister der Länder in der Bundesrepublik], *Neue Medien und Telekommunikation im Bildungswesen. Beschluss der Kultusministerkonferenz vom 28.02.1997*; Heidi Schelhowe u.a., *Medienbildung für die Persönlichkeitsentwicklung, für die gesellschaftliche Teilhabe und für die Entwicklung von Ausbildungs- und Erwerbsfähigkeit*, Berlin: BMBF 2009; Torsten Brinda u.a., *Dagstuhl-Erklärung: Bildung in der digitalen vernetzten Welt*, März 2016; Tom Braun u.a., „Positionspapier zur Weiterentwicklung der KMK-Strategie ‚Bildung in der digitalen Welt‘“, in: *MedienPädagogik: Zeitschrift für Theorie und Praxis der Medienbildung* 2021, S. 1–7.

Verständnis von Bildung in der digitalen Welt,⁹ welches komplexe individuelle und überindividuelle Aushandlungen, mediale Praktiken und digitale Medienkulturen weitestgehend und zugunsten einer rein instrumentellen Perspektive auf digitale Medien ausklammert. In der Folge werden utilitaristische Ansätze zur Entwicklung zeitgemäßer formaler Lern- und Bildungssettings verfolgt, um digitale Kompetenzen im Anschluss an die digitalen Technologien zu fördern.

Die Digitalisierung von Bildung durch Technisierung und den Einsatz von Bildungstechnologien erfährt insbesondere vor dem Hintergrund des bundesweiten Lehrkräftemangels eine neue Qualität, denn hier werden Argumente laut, die abermals auf die Optimierung von Lernerfahrungen abzielen und individuelle Lernbegleitung durch Automatisierung und Berechnung versprechen. Die Gestaltung von digitalen Infrastrukturen und die Potenziale zur individuellen Selbstverwirklichung bleiben damit jedoch meist auf der Oberfläche der uns anstrahlenden Displays. Mit steigender Durchdringung digitaler Technologien wird jedoch auch der dekonstruierende Blick auf die digitalen Technologien immer bedeutsamer, denn wenn das Ziel einer outputorientierten Bildung die Befähigung zur Teilhabe der Menschen in der digitalen Welt ist, dann bildet der Fokus auf das Programmieren und Coding einen Zugang, um das Verständnis über Prozesse einer *black box society*¹⁰ anzustoßen. Nimmt man diese Entwicklungen zusammen, dann ist Programmierung oder Coding nicht nur für bildungspraktische Fragen relevant, sondern auch für die erziehungswissenschaftliche Theoriebildung bedeutsam. Dabei ist bislang weitestgehend unklar, wie sich digitale Technologien in

⁹ Benjamin Jörissen und Dan Verständig, „Code, Software und Subjekt: Zur Relevanz der Critical Software Studies für ein nicht-reduktionistisches Verständnis ‚digitaler Bildung‘“, in: Ralf Biermann und Dan Verständig (Hg.), *Das umkämpfte Netz*, Wiesbaden: Springer 2017, S. 37–50.

¹⁰ Frank Pasquale, *The Black Box Society: The Secret Algorithms That Control Money and Information*, Cambridge, MA: Harvard University Press 2015.

Prozesse der Artikulation und Selbstbestimmung einschreiben. Gleichzeitig scheint ein tiefergehendes Verständnis über die kulturellen Einschreibungen des digitalen Wandels von besonderer Bedeutung, um schließlich überhaupt eine gestaltende Rolle einnehmen zu können.

Dieser Beitrag wird daher nicht explizit auf die Notwendigkeit zur Förderung von Programmierkenntnissen in unterschiedlichen formalen Bildungskontexten abzielen. Es geht auch nicht in erster Linie darum, welche Rolle KI bei der Gestaltung von Lehr-Lern-Settings spielt.¹¹ Es geht vielmehr um die in Copilot eingeschriebenen Lern- und Bildungspotenziale, die sich auch abseits des funktionalistischen Programmierens verorten lassen. Diese Perspektive rückt die Performativität von Code in den Mittelpunkt und lässt auch subjektivierungstheoretische Überlegungen zu. Code wird damit nicht nur hinsichtlich seiner Funktionalität, sondern eingebettet in kulturelle, soziale, aber auch politische Verflechtungen im Sinne der Critical Code Studies (CCS) betrachtet.¹²

Diese Perspektivsetzung erfolgt in drei Schritten. Zunächst soll ein historischer Blick entwickelt werden, um einerseits den kulturtheoretischen Bedeutungsgehalt von Programmcode und Programmieren in historischer Perspektive einzuholen und andererseits die vielschichtigen Herausforderungen bei der Lösung von Problemen im Kontext der Programmierung und der Arbeit mit dem Computer zu adressieren. Darauf aufbauend werden Praktiken des Programmierens hinsichtlich ihrer subjektivierenden Kraft befragt. Die Performativität des Codes spielt dabei eine entscheidende Rolle und ist zugleich ein wesentliches Distinktionskriterium, wenn es um das

¹¹ Vgl. hierzu ausführlich Brett A. Becker u.a., „Programming Is Hard -- Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation“, in: *arXiv*, Dezember 2022, DOI: 10.48550/arXiv.2212.01020.

¹² Mark C. Marino, *Critical Code Studies: Initial Methods*, Cambridge, MA: The MIT Press 2020.

Schreiben mit der Maschine geht. Code ist nicht einfach nur Text, er wird in Maschinensprache übersetzt, ausgeführt und ist damit emergent, obgleich er strikt regelgeleitet ist. Code ist in seiner Beschaffenheit und unter den Bedingungen der Produktion und Ausführbarkeit hochgradig komplex und abhängig von Materialität wie die Hardware, auf der er läuft, den sozialen Gefügen, unter denen er entstanden ist, und welche Werte und Weltansichten implementiert sind oder nicht. Im dritten Teil wird der Blick nach vorn gerichtet. Es geht abschließend darum, die Grenzen und Reichweiten von Automatisierung im Kontext der Programmierung zu bestimmen und ein interdisziplinäres Forschungsfeld zu konturieren, welches durch eine hohe Dynamik gekennzeichnet ist und weiter reicht als die rein funktionalistische Betrachtung von Code.

2. Ein Blick zurück: Zur Bedeutung von Programmcode und Problemlösung

Am 9. September 1947 stieß die Informatikerin und Konteradmiralin der U.S. Navy, Grace Hopper, bei ihrer Arbeit am Harvard Mark II-Computer auf Unregelmäßigkeiten. Es hat sich sprichwörtlich ein *bug* am fast 25 Tonnen schweren Computer eingeschlichen, denn als Hopper und ihre Kollegen Teile der Verkleidung öffneten, um den Unregelmäßigkeiten auf die Spur zu kommen, stellten sie fest, dass eine Motte in ein Relais geflogen war. Nachdem die Störung ausfindig gemacht wurde, klebte Grace Hopper die berüchtigte Motte in das Betriebsprotokoll mit dem Kommentar „First actual case of bug being found“.¹³

¹³ Vgl. auch P.A. Kidwell, „Stalking the Elusive Computer Bug“, in: *IEEE Annals of the History of Computing* 20:4 (1998), S. 5–9.

Der Mark II wie auch schon der Mark I waren keine Computer mit gespeicherten Programmen; es wurde nur eine Anweisung des Programms von einem Band ausgelesen und ausgeführt. Diese Trennung von Daten und Anweisungen wird als Harvard-Architektur bezeichnet. Das Band mit dem Programminhalt konnte dabei nur acht Befehle aufnehmen. Jede Sekunde war in mehrere Perioden unterteilt. Eine Anweisung konnte je nach Periode auch unterschiedliche Bedeutungen haben. Grace Hopper war neben den Projekten am Mark I und Mark II auch am UNIVAC beteiligt. Es war der erste Großrechner, der in Serie gebaut wurde. 1952 entwickelte Hopper hierfür den ersten Compiler und setzte sich darüber hinaus beharrlich dafür ein, dass die Programmierung in für Menschen allgemeinverständlichen Sprachen erfolgen sollte. In diesem Zusammenhang entwickelten Hopper und ihr Team auch die Programmiersprache FLOW-MATIC¹⁴ und legten damit den Grundstein für die Programmiersprache COBOL. Mark C. Marino diskutiert diese Entwicklungslinie vor dem Hintergrund der Frage, inwiefern natürliche Sprache das kritische Lesen von Code ermöglicht oder erschwert. Einerseits wird entlang dieser Diskussion nochmals analytisch hervorgehoben, dass das Lesen und Schreiben von Code in der Programmierung anderen Regeln folgt als das natürliche Lesen und Schreiben. Andererseits, so arbeitet Marino heraus, zeigt sich an FLOW-MATIC, wie sich Einschreibungen der englischen Sprache in Programmierweisen und -paradigmen manifestiert haben und wie sich Differenzlinien zum Verstehen von Befehlen und der Funktion von Code abzeichnen.¹⁵ Natürliche Sprache bildet insofern einen wichtigen Ankerpunkt, da Code von Menschen für Menschen geschrieben wird. Die kulturhistorische Leistung der von Marino vorgelegten Auseinandersetzung

¹⁴ Vgl. Marino, *Critical Code Studies*, S. 129ff.

¹⁵ Vgl. Ebd., S. 36.

ist jedoch nicht nur die linguistische Perspektive auf das Verhältnis von Maschinensprache und natürlicher Sprache, sondern auch die Reflexion darüber, wie sich die Kluft zwischen den Geschlechtern in den darauffolgenden Jahren in professionellen Programmierkulturen vergrößert hat.

Computer wie der Mark II füllten damals mit ihrer Architektur ganze Räume aus, sie erzeugten Wärme und zogen damit nicht zuletzt auch Insekten an, die schließlich auch auf die mechanischen Schaltungen eingewirkt haben. Auch wenn die Bedeutung des englischen Begriffs *bug* als Fehler nicht allein auf den Vorfall von 1947 zurückzuführen ist, hat er in der Softwareentwicklung und der Informatik bis heute Bestand und einen besonderen Stellenwert. Während es sich bei der Entdeckung der Motte durch Grace Hopper noch um ein Problem an der Hardware handelte, gilt der Ausdruck *bug* inzwischen allgemein als Synonym für technische Fehler in Computerprogrammen. Die Lösung eines Softwareproblems wird heutzutage als Bugfixing bezeichnet. Es handelt sich dabei um einen wichtigen Bestandteil in der Programmierung, bei dem Entwicklungsumgebungen selbst nicht nur strukturiert sind, sondern eine strukturierende Funktion haben, indem sie Syntaxfehler hervorheben, Hinweise und Vorschläge zu Programmierrichtlinien anzeigen und bei komplexen Anwendungen ganze Debugger-Tools in die IDE integriert sind. Daneben kommen heute auch Profiler zum Einsatz. Dabei handelt es sich um Computerprogramme, die ein zu untersuchendes Programm während des Ausführens analysieren und Informationen darüber bereitstellen, welche Funktionen zur Laufzeit wie viel Zeit benötigen.

Schon die frühen Jahre des Computers und der Programmierung waren geprägt von pragmatischen und teilweise unkonventionellen Problemlösungsansätzen und Bestrebungen zur Dokumentation des Erfahrungswissens, welches in der Arbeit an den Rechnern entstanden ist, beginnend mit Hop-

pers ins Protokoll geklebten Motte. Früher wurden Maschinenbefehle auf unterschiedlichen Medien oftmals mühsam manuell erstellt und dann der Maschine zur Verarbeitung vorgelegt. Heutzutage werden Befehle in Entwicklungsumgebungen getippt oder von automatisierten Systemen gar vorgeschlagen, die oft online gehostet werden. Man erhält dabei nahezu sofort, also *on the fly*, eine Rückmeldung über die Kompilierungs-/ Testergebnisse der letzten Änderungen und ihrer Funktionalität. Neben der steigenden Vernetzung lässt sich hier vor allem eine Komplexitätssteigerung durch die Automatisierung von Abläufen festmachen. Programmieren setzt heute nach wie vor noch ein hohes Maß an Flexibilität und Problemlösungskompetenz voraus, doch die Rahmenbedingungen, unter welchen programmiert wird, haben sich grundlegend gewandelt. KI-basierte Systeme nehmen heute nicht nur die Rolle von Vorschlagemechanismen ein, es wird von den dahinterstehenden Unternehmen versprochen, dass diese KI-Systeme vollwertige Teammitglieder im Prozess der Programmierung werden können. Damit gibt es nicht nur Veränderungen beim Programmieren, es wandeln sich womöglich auch Rollenverständnisse. Zudem werden Veränderungen deutlich, die das Erlernen von Programmierkenntnissen und die didaktische Gestaltung von Programmierkursen betreffen. Die Motte mag noch im System sein, doch sie zu finden, wird angesichts steigender Komplexität und Undurchsichtigkeit womöglich zu einer neuen Herausforderung.

3. Ein Blick über die Schulter: Codex und Copilot

Der GitHub Copilot wurde in Partnerschaft der US-amerikanischen Unternehmen OpenAI und Microsoft entwickelt und im Juni 2022 für die individuelle Nutzung und Entwicklung veröffentlicht. Das Tool richtet sich gleichermaßen an

Anfänger:innen und Expert:innen. Der Copilot soll dazu beitragen, dass Entwickler:innen ihren Code effizienter schreiben können. Effizienz wird hierbei in erster Linie entlang der Faktoren Zeitaufwand und Sicherheit definiert. Die Effizienzsteigerung soll auf Basis von KI erfolgen, indem Vorschläge für Codezeilen, ganze Codeblöcke und komplexe Algorithmen basierend auf dem jeweiligen Entwicklungskontext vorgeschlagen werden. Dadurch sparen die Entwickler:innen wertvolle Zeit beim Schreiben von Code unterschiedlicher Komplexität. Dabei wird auf Programmcode zurückgegriffen, der unter den offenen Repositorien auf GitHub verfügbar ist. Das KI-Modell der ersten Version von Copilot wurde mit Codex trainiert. Codex basiert zu seiner Veröffentlichung auf GPT-3 (Generative Pre-trained Transformer der dritten Generation). Die Trainingsdaten wurden im Mai 2020 gesammelt und umfassen mehr als 54 Millionen GitHub-Repositorien, darunter waren mit 179 GB überwiegend Projekte in der populären Programmiersprache Python. Herausgefiltert wurden Dateien, bei denen die Wahrscheinlichkeit hoch ist, dass sie automatisch generiert wurden, sowie solche mit mehr als durchschnittlich 100 Zeilen Code, einer maximalen Zeilenlänge von mehr als 1000 und solchen, die einen geringen Anteil alphanumerischer Zeichen enthalten, sodass die Trainingsdaten schließlich 159 GB umfassten.¹⁶ Codex kann englischsprachige Befehle (*prompts*) entgegennehmen und dann Code in mehreren Programmiersprachen generieren, darunter JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript und Shell. Es ist auf Basis der Trainingsdaten am leistungsfähigsten, wenn Python-Code generiert werden soll. Die Codex API wird jedoch seit dem 23. März 2023 nicht mehr unterstützt, da schon GPT-3.5 wesentlich leistungsfähiger sei und GPT-4 eine wesentlich umfassenden

¹⁶ Mark Chen u.a., „Evaluating Large Language Models Trained on Code“, in: *arXiv*, Juli 2021, DOI: 10.48550/arXiv.2107.03374.

dere Basis darstellen soll. Auch hieran zeigt sich eine gewisse Flüchtigkeit und die hohe Dynamik in der Entwicklung. Codex wurde ursprünglich als kostenlose, begrenzte Beta-Version im Jahr 2021 eingeführt und wurde dann schließlich von einem neueren Produkt abgelöst.

Schon bei der ersten Version von Copilot ist es bemerkenswert, dass Code auch zwischen unterschiedlichen Programmiersprachen übersetzt werden kann. Die Funktionalität des als Eingabe bereitgestellten Codes (auf Englisch) kann erläutert und die Zeitkomplexität des generierten Codes ermittelt werden. Damit hat schon die erste Fassung eine dialogische Komponente und bringt eine spezifische Strukturierung von Erklärbarkeit mit sich. Im Frühjahr 2023 hat GitHub eine verbesserte Version von Copilot veröffentlicht und diese Copilot X genannt. In der aktualisierten Version habe man nicht nur das aktuelle GPT-4-Modell von OpenAI übernommen, der KI-Assistent verfüge unter anderem nun auch über ein Chatfenster direkt im Editor. Zudem gibt es neue Einsatzfelder wie Pull Requests und die Kommandozeile. Diese raschen Entwicklungen sind ein Signum für die hohe Dynamik, die sich durch den Einsatz von KI in der Programmierung beobachten lässt. Mit der Veröffentlichung von Copilot X zeigt sich, dass hier gerade die Chatfunktion eine Unterstützung sein kann, um Unklarheiten zu erläutern und Leerstellen im Verstehensprozess zu minimieren oder gar auszuräumen. Man kann nicht nur Abläufe und Prozesse erklären, sondern über den Chatdialog auch spezifische Unit-Tests durchführen und so den Code mit unterschiedlichen Parametern live erproben. Dies ist auch über Spracherkennung möglich. Mit Copilot Voice kann man direkt mit dem Assistenzsystem sprechen und so interagieren. Allein durch diese Möglichkeiten verändert sich die individuelle Programmiererfahrung, es entstehen neue Routinen und es bilden sich womöglich neue Konventionen in der Art und Weise, wie sich Praktiken des Programmierens und des niederschweligen Codings voll-

ziehen. Copilot X erfasst Fehler direkt im Code und bietet Lösungsvorschläge sowie Erklärungen an. Zudem wird die Zusammenarbeit bei umfangreichen Änderungen auf GitHub umfassender unterstützt. Dies geschieht, indem der Code vom KI-System analysiert wird und bei umfangreichen Pull Requests beispielsweise die Beschreibung der Neuerungen automatisiert verfasst wird. Damit geht Copilot X weit über die IDE hinaus und birgt durch die Erläuterungen und die implementierten Strukturen zur Nachvollziehbarkeit der teils automatisierten Prozesse auch Lernpotenziale. Erklärungen werden dann nicht mehr direkt von Menschen gegeben, sondern von der Maschine aus großen Datenmengen errechnet und für Menschen übersetzt. Dies schlägt sich auch auf Programmierkonzepte nieder, bei denen Menschen direkt am Code zusammenarbeiten, denn mit den jüngsten Neuerungen verspricht das Unternehmen eine noch umfassendere Integration der KI als vollwertiges Teammitglied, mit dem man sich über neuere Pull Requests oder Codeänderungen sprichwörtlich unterhalten könne. Der Copilot und die vom System produzierten Daten bieten damit auch einen methodischen Zugang zu Quellcodekritik, indem Erklärungen zur Funktionsweise von Code angeboten und protokolliert werden.

Der Copilot wird von GitHub mit dem Slogan „Your AI pair programmer“ präsentiert. Damit spielt das Unternehmen auf die in der Softwareentwicklung etablierte Methode der Paarprogrammierung an. Dabei handelt es sich um eine Praktik, bei der zwei Entwickler:innen gemeinsam an einem Projekt arbeiten und, in Anlehnung an die Luftfahrt, die Entwickler:innen in die Rollen von Pilot:in und Copilot:in schlüpfen. Eine Person schreibt, die andere Person überblickt den Prozess. Das *code review*, also die Überprüfung des geschriebenen Programmcodes, erfolgt damit *on the fly* beim Mitlesen und Mitdenken sowie dem Blick über die Schulter durch die Copilotin oder den Copiloten. Man wirkt somit der Herausbildung von Wis-

sensinseln in Programmerteams entgegen und fördert zugleich den sozialen Austausch, da die gemeinsame Programmierung den Prozess der Kommunikation expliziert. *Pair programming* kann auch remote, also unter den Bedingungen der Telepräsenz erfolgen. Durch Projektmanagement-Tools wie Slack und Screensharing besteht schließlich nicht erst seit der Pandemie die Möglichkeit zur räumlich unabhängigen Zusammenarbeit. Mit Copilot wird nun jedoch das Angebot gemacht, diese Strukturen aufzubrechen und Entwicklungssettings neu zu strukturieren. Dadurch wird nicht nur das Schreiben von Code restrukturiert, sondern auch grundlegend in das Setting der Programmierung und schließlich auch in die Zusammensetzung von Programmerteams und Routinen der Wissensarbeit eingegriffen.

Schon die frühe Version von Copilot ist Gegenstand verschiedener Untersuchungen geworden, so wurden beispielsweise Sicherheitsaspekte,¹⁷ Erfahrungen und Einstellungen von Entwickler:innen¹⁸ erforscht und didaktische Implikationen im Kontext der Lehre¹⁹ sowie die pädagogische und akademische Integrität aus informatischer Perspektive diskutiert.²⁰ Bislang bleibt jedoch zu klären, inwiefern sich die technologische Entwicklung auch in Artikulationsprozesse beim Programmieren einschreibt und wie die Interpretation von Quellcode hierdurch beeinflusst wird. Nachfolgend sollen zunächst einige Aspekte und Herausforderungen in der didaktischen Gestaltung von Lernsettings adressiert werden, um so schließlich auf die subjekttheoretischen Bedeutungen hinzuweisen,

¹⁷ Jingxuan He und Martin Vechev, „Controlling Large Language Models to Generate Secure and Vulnerable Code“, in: *arXiv*, Februar 2023, DOI: 10.48550/arXiv.2302.05319.

¹⁸ Mateusz Jaworski und Dariusz Piotrkowski, „Study of Software Developers’ Experience Using the Github Copilot Tool in the Software Development Process“, in: *arXiv*, Januar 2023, DOI: 10.48550/arXiv.2301.04991.

¹⁹ Vgl. Becker u.a., „Programming Is Hard“.

²⁰ Michel Wermelinger, „Using GitHub Copilot to Solve Simple Programming Problems“, 2023.

die sich beim Programmieren mit dem Copilot einschreiben und aufgrund ihrer Komplexität bislang ein Forschungsdesiderat sind.

4. Ein Blick auf die Lern- und Bildungspotenziale im Umgang mit Copilot

Brett A. Becker u.a. haben sich aus didaktischer Perspektive mit Codex und Copilot befasst und untersucht, welche Auswirkungen der Copilot auf Einführungskurse in der Informatik und das Erlernen von grundlegenden Programmierkenntnissen hat. In ihrem Positionspapier argumentieren sie für eine rasche, zugleich jedoch kritisch-reflexive Reichweitenabschätzung der neuen Technologie bei der didaktischen Gestaltung von Programmierkursen. Ohne schnelle, gezielte und konzertierte Anstrengungen verlieren Pädagog:innen ihren Vorteil der sich bietenden Möglichkeiten in der Konzeption und Umsetzung von zeitgemäßen Lern- und Bildungssettings, die auch KI produktiv einbeziehen und somit eine Förderung von grundlegender KI-Bildung ermöglichen.²¹ Das Codex-Modell hat sich bei der Lösung von Programmieraufgaben, die in einfachem Englisch präsentiert wurden, als sehr leistungsfähig erwiesen. Auch komplexere Problemstellungen konnten von Codex gelöst werden. Das ist für die Ausbildung und Einführungskurse in die Informatik nicht unwesentlich, denn die ersten Aufgabenstellungen sind meist auf Syntax und Prinzipien der Programmierung ausgelegt und es dauert einige Zeit, bis die Studierenden diese Grundlagen beherrschen und unterschiedliche Programmierparadigmen erlernt haben. Wenn nun KI-Modelle zur Produktion von Code verwendet werden können, um die Aufgaben zur Implementierung beispielsweise von Sor-

²¹ Vgl. Becker u.a., „Programming Is Hard“.

tialgorithmen zu lösen, können sich die Studierenden zwar schneller auf komplexere Algorithmen fokussieren, es können aber auch Leerstellen bei grundlegenden Zusammenhängen entstehen, die sich später nicht nur auf die Qualität der Problemlösung auswirken, sondern auch motivationale Aspekte betreffen, wenn die Lernkurve plötzlich zu steil ist.

Für Becker u.a. birgt der Einsatz von Copilot einige Chancen für den Aufbau und die Pflege von pädagogischen Beziehungen bei der Gestaltung von Einführungskursen zur Programmierung.²² Sie sehen in erster Linie den Bedarf, das Lesen und Bewerten von Code anstelle der Produktion von Code in den Mittelpunkt zu stellen. Es gehe dann auch darum, ein ethisches Bewusstsein für unterschiedliche Problemstellungen und eine Haltung zu entwickeln und nicht einfach nur um die Implementation von Codebeispielen. Diese Position lässt sich im Einklang zu der von Annette Vee entwickelten Konzeption von *coding literacy* beschreiben, bei der es neben funktionalistischen Aspekten von Code vor allem um soziale und kommunikative Kontexte geht.²³ Hieraus ergibt sich eine doppelte Differenz, die sich einerseits in der Herstellung einer Balance von Freiheit und Instruktion beschreiben lässt und andererseits auf die individuellen und situativen Erfahrungen der Lernenden abhebt, in denen die Performativität des Codes erfahr- und explizierbar gemacht werden kann. Der von Copilot vorgeschlagene Code ist nicht immer vollständig und muss gegebenenfalls ergänzt werden. Wie diese Ergänzungen erfolgen, ist bislang noch weitestgehend ungeklärt. Bei Unit-Tests beispielsweise werden verschiedene *asserts* vorgeschlagen, die Entscheidung, wie man damit umgehen möchte, liegt dann beim Menschen. Hierfür ist Vorwissen über die Abläufe von Unit-Tests und deren konzeptionelle Bedeutung sowie Kontexte wichtig. Wann Unit-Tests

²² Ebd.

²³ Vee, *Coding Literacy*.

sinnvoll sind, wie viele *asserts* beispielsweise hilfreich erscheinen, muss dann individuell entschieden werden.

Mark Chen u.a. machen auf die Herausforderung aufmerksam, dass die Verwendung von Modellen zur maschinellen Erstellung von Programmcode dazu führen kann, dass man sich zu sehr auf die generierten Vorschläge verlässt.²⁴ Gerade Noviz:innen, die solche Modelle verwenden, können sich schnell an die automatisch vorgeschlagenen Lösungen gewöhnen. Dieser Effekt könne dann noch verstärkt werden, wenn Werkzeuge zur Unterstützung direkt in eine Entwicklungsumgebung eingebettet sind, wie dies bei Copilot der Fall ist. Studierende können dann dazu neigen, die Problemstellung nicht sorgfältig oder überhaupt nicht zu lesen, und denken dann nicht mehr über die Rechenschritte nach, die zur Lösung eines Problems erforderlich sind. Diese Herausforderungen lassen sich nicht nur bei Noviz:innen mit grundlegenden Programmierkenntnissen herausstellen, sie betreffen auch erfahrene Entwickler:innen. Es geht in lerntheoretischer Perspektive nicht nur um das Erlernen, sondern auch um das Umlernen, wenn bestehende Routinen in der Softwareentwicklung infrage gestellt werden oder durch beispielsweise Chatinterfaces verändert werden können, dann werden auch bestehende Ordnungen zur Disposition gestellt. Bei der Programmierung korrespondiert diese potenzielle Diskontinuität mit dem bereits erworbenen Wissen über Programmierparadigmen, Entwicklungsumgebungen und organisationale Strukturen. Es geht dann schließlich weniger um das Erlernen eines Gegenstands, sondern vielmehr um die Flexibilität, das Lernen zu erlernen.²⁵

Fasst man das Programmieren selbst als performativen Akt, dann stellt sich in Anlehnung an Geoff Cox und Christopher Alex McLean die Frage danach, ob bzw. in welchem Maße man

²⁴ Mark Chen u.a., „Evaluating Large Language Models Trained on Code“.

²⁵ Käte Meyer-Drawe, „Lernen als Erfahrung“, in: *Zeitschrift für Erziehungswissenschaft* 6:4 (2003), S. 505–514.

selbst programmiert oder programmiert *wird*.²⁶ Die Maschine erwartet Befehle und fordert zur Eingabe auf. Damit gibt die Maschine ganz direkt Befehle und Ausführungslogiken vor, die dann in der Interaktion mit der Maschine vom Menschen (re-)strukturiert werden, um eine spezifische Ordnung, also auch implizite Wertevorstellungen abzubilden.²⁷ In der Praxis ergeben sich gerade bei Code-Vorschlägen, die *on the fly* erfolgen, einige performative Verflechtungen, die sich als Wechselverhältnis von Aufforderung und Befehlseingabe beschreiben lassen: Aus jedem generierten Vorschlag und jeder manuell getippten Eingabe entsteht Zeichen für Zeichen ein Wechselverhältnis zwischen Mensch und Maschine, bei dem die Rollen von Aufforderung und Befehlseingabe zwischen beiden oszillieren. Während bei automatischen Vorschlägen die Irritation und Widerständigkeit steigt und der flüssige Prozess des Schreibens von Code damit unterbrochen wird, ergeben sich Momente des *flows* durch affirmative und womöglich gut antizipierte oder berechnete Vorschläge. Genau in diesem Spannungsfeld entstehen Verdeckungen von Problemlösungsstrategien und der Prozess der Aushandlung über die Passfähigkeit von geschriebenem Code wird so nicht mehr expliziert. Es bleibt bei einer impliziten und womöglich verdeckten Einschreibung, bei denen die Aushandlungen über den Lösungs- oder Implementierungsweg zunächst zwischen Mensch und Maschine vollzogen werden und die Identifikation von Problemstellungen direkt in die Überwachung durch Copilot fließt. Es erfordert also Vertrauen, sich auf die Technologie einzulassen. Zugleich ist ein Bewusstsein über die Unbestimmtheit, die in den berechneten Vorschlägen liegt, ebenso nötig, wie das

²⁶ Vgl. Geoff Cox und Christopher Alex McLean, *Speaking Code: Coding as Aesthetic and Political Expression*, Cambridge, MA: The MIT Press 2012, S. 85.

²⁷ Vgl. hierzu Dan Verständig und Juliane Ahlborn, „Decoding Subjects? Über Subjektivierung und Kreativität im algorithmischen Zeitalter“, in: Jens Holze, Dan Verständig und Ralf Biermann (Hg.), *Medienbildung zwischen Subjektivität und Kollektivität: Reflexionen im Kontext des digitalen Zeitalters*, Wiesbaden: Springer 2020, S. 89.

Wissen über die stetige Korruption der noch so elegantesten Lösung durch statistische Berechnung.

Copilot kann nach Michel Wermelinger einen ersten hilfreichen Ansatz zu einem Problem liefern.²⁸ Die Lernenden müssen jedoch die Syntax und Semantik einer Programmiersprache gut kennen, um weniger passende oder gar falsche Vorschläge zu erkennen und zu ändern. Auch hier zeigt sich die Problematik der Verdeckung von Problemlagen und die Herausforderung zur Identifikation dessen, was überhaupt problematisch sein könnte. Es geht dann weniger um rein technische Probleme, sondern auch um Technikfolgeabschätzungen und ethische Fragen, die sich auch in der Herausbildung einer Haltung als Entwickler:in manifestieren. Dies wird auch von Becker u.a. diskutiert. Eine Haltung zu den Dingen der Welt und zu sich selbst zu entwickeln, ist im Kern ein erziehungswissenschaftlicher Topos, der sich als Differenzbewusstsein beziehungsweise Orientierungswissen umreißen lässt.²⁹ Zu wissen, was in welchen Situationen richtig oder falsch ist, erfordert jedoch Reflexionsanlässe und Zeit der Verarbeitung. Dementsprechend kann davon ausgegangen werden, dass die rein affirmative Nutzung, die Variation und Friktion ausklammert, einer kritisch-reflexiven Haltung entgegenwirken könnte. Dementsprechend erscheint es mit Blick auf die Lern- und Bildungspotenziale von GitHub sinnvoll, didaktische Aspekte mit einer bildungstheoretischen Perspektive zu verknüpfen, denn es geht dann weniger um die Momente, in denen Handlungssicherheit durch Bestimmtheit hergestellt wird, sondern vor

²⁸ Michel Wermelinger, „Using GitHub Copilot to Solve Simple Programming Problems“.

²⁹ Vgl. hierzu Jürgen Mittelstraß, „Bildung und ethische Maße“, in: Nelson Killius (Hg.), *Die Zukunft der Bildung*, Frankfurt a.M.: Suhrkamp 2002, S. 151–170. Zum Bildungsbegriff auch Winfried Marotzki und Benjamin Jörissen, „Wissen, Artikulation und Biographie: Theoretische Aspekte einer strukturalen Medienbildung“, in: Johannes Fromme and Werner Sesink (Hg.), *Pädagogische Medientheorie*, Wiesbaden: Springer 2008, S. 51–70.

allem auch um jene Figurationen, in denen Handlungsrou-
tinen infrage gestellt werden und Unbestimmtheit erkannt
beziehungsweise anerkannt wird. Diese Momente der Irrita-
tion können nicht nur als Lernen, sondern vor dem Hinter-
grund ethischer Dimensionen und Grenzen auch als Bildungs-
anlässe beschrieben werden.³⁰ Es geht dann um das Verhältnis
zur Herstellung von Handlungssicherheit bei gleichzeitiger
Anerkennung von Unbestimmtheit. Dieses Verhältnis erinnert
an die klassische Subjektphilosophie, die bei aller Vielseitigkeit
ihrer Analysen eine Gemeinsamkeit entwickelt hat – die Auto-
nomie des Subjekts, als ein sich im Handeln erkennendes, wel-
ches auf sich selbst gerichtet ist und nicht (allein) die Kontin-
genz der Weltbezüge verhandelt. Das Subjekt erscheint in die-
sen Figurationen als Selbst, als expressiver Kern der
Selbstverwirklichung.³¹ Ihm werden damit universale Eigen-
schaften zugeschrieben, die sich auch in den großen Erzählun-
gen der Moderne entlang der emanzipatorischen Kraft von
autonomer, transparenter und kreativer Subjektivität finden.³²
Diese klassische Konzeption des Subjekts wird begleitet von
Kritik, die sich unter anderem in der Dezentrierung des Sub-
jekts niederschlägt. Das Subjekt, wenn es nicht gar Fiktion ist,
wird in dieser Lesart von gesellschaftlich-kulturellen Struktu-
ren abhängig konzipiert. Dabei werden die Ambivalenzen der
Subjektformen und die Widersprüchlichkeiten relevant, die
sich entgegen den Ordnungen von Autonomie und Transpa-
renz verorten lassen. Für die Auseinandersetzung mit KI in der
Programmierung und eine erziehungswissenschaftliche Ana-
lyse sind besonders diese Widersprüchlichkeiten von gestei-

³⁰ Winfried Marotzki, *Entwurf einer strukturalen Bildungstheorie: Biographietheoretische Auslegung von Bildungsprozessen in hochkomplexen Gesellschaften. Studien zur Philosophie und Theorie der Bildung*, Bd. 3, Weinheim: Deutscher Studien Verlag 1990.

³¹ Andreas Reckwitz, *Subjekt. Einsichten – Themen der Soziologie*, Bielefeld: Transcript 2012, S. 13.

³² Vgl. Elisabeth Sattler, *Die riskierte Souveränität. Erziehungswissenschaftliche Studien zur modernen Subjektivität*, Bielefeld: Transcript 2009, S. 9f.

gerter Bedeutung, denn wenn es kein Subjekt ohne Subjektivierung geben kann, dann stellt sich die Frage nach den Formen des machtvollen Unterworfenseins und damit der Art und Weise, wie sich Subjektivierung³³ vollzieht.

Schon Donald Knuth wies 1968 darauf hin, dass das Programmieren nicht nur einen wirtschaftlichen oder wissenschaftlichen Mehrwert hat, sondern auch hinsichtlich der ästhetischen Erfahrung und des Selbstaudrucks bedeutsam ist.³⁴ Knuth zieht hier Vergleiche zum Verfassen von Poesie oder Komponieren von Musik. Programmieren ist für ihn mit dem Anspruch an künstlerischen Ausdruck verbunden, der dazu auffordert, Algorithmen und Code in seiner Beschaffenheit zu hinterfragen und sich zu ihnen zu verhalten. Mit einer solchen Perspektivierung geht das Bewusstsein einher, sich über den Code auszudrücken. Diese Form des Ausdrucks beherbergt einiges an Kreativitätspotenzial beispielsweise dann, wenn eine künstlerische Handschrift beim Schreiben von Code entwickelt wird. Gleichzeitig ist der Ausdruck voraussetzungsvoll, wie hinsichtlich der lerntheoretischen Perspektive bei Becker u.a. bereits angedeutet wurde. Eine wie auch immer gelagerte Virtuosität ergibt sich nicht von selbst, sondern geht aus Erfahrungen, Fehlern und Suchbewegungen hervor, die sich auf individueller Ebene abzeichnen und zugleich auf die (soziale) Welt abstellen. Der Copilot tritt hier als eine Entität in das Ensemble der ästhetischen Expression, indem er einerseits auf eine Vielzahl von Material zurückgreift, um Vorschläge beim Coding zu unterbreiten, und damit andererseits in den Dialog zum individuellen Ausdruck der Entwickler:innen tritt.

Formen des Ausdrucks lassen sich insbesondere dann besser bestimmen, wenn über die rein funktionalistische Problemlö-

³³ Judith Butler, *Haß spricht: zur Politik des Performativen*, Frankfurt a.M.: Suhrkamp 2006, S. 15f.

³⁴ Donald E. Knuth, *The Art of Computer Programming*, Reading, MA: Addison-Wesley 1968.

sung hinausgeschaut wird. Folgt man Susanne Grabowski und Frieder Nake, so ist der Ausdruck über den Computer immer ein Prozess, der mit Dingen aus der Ferne umgeht.³⁵ Man hat es hier also immer mit Abstraktion zu tun. Eine Linie zeichnet man beispielsweise nicht einfach auf eine Leinwand, man verbindet stattdessen zwei Punkte im Koordinatensystem. Diese Ausdrucksformen können als algorithmische Artikulation gefasst werden. Sie zeichnen sich dadurch aus, dass sie „[...] einerseits in ihrer Strukturiertheit bedingt durch die Materialitäten, die sie hervorbringen können, sowie andererseits in ihrer Ausführbarkeit und damit hinsichtlich ihrer Performativität charakterisiert sind“.³⁶ Algorithmische Artikulation lässt sich grundlegend hinsichtlich ihrer „Dynamisierung, Performativität und Intransparenz“ festmachen³⁷ und somit auch von anderen Formen (medialer) Artikulation abgrenzen. Der GitHub Copilot, aber auch andere KI-basierte Systeme schreiben sich auf unterschiedliche Weise in die Artikulationsweisen der Menschen ein und können als eine komplexe wechselseitige Beziehung der Interaktion von Mensch und Maschine gesehen werden. Es werden die Grenzen zwischen Werkzeug und der Maschine aufgebrochen, wenn die performative Qualität von Code zur Entfaltung kommt und Leerstellen auch in einem spekulativen Sinne adressiert werden.³⁸ Das Spiel mit der Technologie steht dann nicht mehr der funktionalistischen Programmierung gegenüber, sondern wird als inhärenter Bestandteil des Codings durch Praktiken der algorithmischen Artikulation sichtbar. Es kommen dann widersprüchliche und

³⁵ Susanne Grabowski und Frieder Nake, „Algorithmische Kunst als Bildungsgegenstand: Gedanken zu einer fachlichen Bildung über Fächer hinaus“, in: *Medienpädagogik: Zeitschrift für Theorie und Praxis der Medienbildung* 33 (2019), S. 76–101.

³⁶ Verständig und Ahlborn, „Decoding Subjects?“, S. 88f.

³⁷ Ebd.

³⁸ Vgl. Geoff Cox, Alex McLean und Adrian Ward, „Coding Praxis: Reconsidering the Aesthetics of Code“, in: Olga Goriunova und Alexander Shulgin (Hg.), *Read_me Software Art and Cultures* 2004, S. 161–174.

potenziell störende Eigenschaften von Code zum Tragen, die sich in der rein funktionalistischen Perspektive auf Programmierung nicht erfassen lassen, jedoch in der künstlerischen Auseinandersetzung einen integralen Bestandteil ausmachen. Es ändern sich dabei nicht nur die Parameter in der Entwicklung, sondern auch die Koordinaten von Bildung, indem erst durch die Grenzgänge oder auch -überschreitungen neue Bildungsräume und ästhetische Lernerfahrungen aufgespannt werden.

5. Ein Blick nach vorn: Fazit und Ausblick

KI-generierter Programmcode ist inzwischen ein fester Bestandteil in der kommerziellen und nicht kommerziellen Softwareentwicklung sowie der Ausbildung. Sowohl für die Didaktik der Informatik als auch für die Medienpädagogik ist bislang noch nicht klar, wie darauf reagiert werden kann, um die Herausforderungen beim Erlernen von Programmierkonzepten zu bewältigen und die Potenziale der KI-Systeme zur Entfaltung zu bringen. Es ist davon auszugehen, dass zukünftig mehr automatisch generierter Code existieren wird. Es ist auch davon auszugehen, dass es hierzu weitere Regulierungsbestrebungen geben wird. Dementsprechend werden insbesondere Lehrpläne der Informatikausbildung auch darauf reagieren müssen. Eine Möglichkeit der Auseinandersetzung haben Becker u.a. beschrieben, indem sie auf die Notwendigkeit zur frühen Reflexion über ethische Fragen in der Informatik abgestellt haben. Dabei sind algorithmisches Denken, das Verständnis der eingesetzten Architekturen, Kompetenzen bei der Fehlersuche und Kommunikationsfähigkeiten nach wie vor relevant. Sie gewinnen vor einem lerntheoretischen Hintergrund sogar an Bedeutung. Mit steigender Zugänglichkeit der Systeme ergeben sich auch neue Anforderungen an eine medi-

enpädagogische Schnittstellenarbeit, denn wenn es mehr Menschen möglich ist, niederschwellig auch komplexere Programmieraufgaben zu formulieren und zu implementieren, dann braucht es auch hier eine zielgruppenorientierte und anlassbezogene Auseinandersetzung. Dies ist eine besondere Chance für den kreativen Einsatz von digitalen Technologien, wie man es heute insbesondere in künstlerisch-explorativen Auseinandersetzungen vorfindet.³⁹

Der Beitrag hat, ausgehend von einer historischen Perspektive auf die Problemlösung und die Bedeutung von Kommunikation in der Entwicklung von Hard- und Software, eine erziehungswissenschaftliche Perspektive auf den Einsatz von KI-Systemen zur Unterstützung bei der Programmierung entfaltet, die sich nicht ausschließlich auf funktionalistisches Programmieren bezieht. Die Betrachtung von historischen, sozialen und konzeptionellen Kontexten des Codings kann dazu beitragen, es als ein wichtiges und kulturell bedeutsames Kommunikationsphänomen zu verstehen und es nicht bloß als eine weitere Kompetenz abzutun. Wenngleich im Beitrag rechtliche Aspekte weitestgehend ausgeblendet wurden, stellen sie nach wie vor eine Herausforderung für den Einsatz von KI im Zusammenhang mit Programmierung dar. Urheberrechtliche Fragen und jene nach der Autor:innenschaft maschinell erstellter Werke werden sicherlich auch in naher Zukunft relevant sein und die technologischen Rahmenbedingungen prägen. Sie rücken ethische Aspekte in den Mittelpunkt und lassen Konfliktlinien hinsichtlich der Performativität von Code deutlich werden, wenn beispielsweise zwischen Quellcode und kompiliertem Code unterschieden wird. Sie sind aber auch dahingehend bedeutsam, als sich das Verständnis der Techno-

³⁹ Vgl. hierzu ausführlich Juliane Ahlborn, „KI – Kunst – Bildung: Wie komplexe algorithmische Systeme das Verhältnis von Kunst, Ästhetik und Bildung verschieben“, in: Christian Leineweber, Max Waldmann und Maik Wunder (Hg.), *Materialität – Digitalisierung – Bildung*, Hagen: Klinkhardt 2023, S. 192–209.

logien im Diskurs durch die Praktiken im Umgang mit diesen Technologien selbst verändert.

Der Beitrag ist ein Plädoyer für die offene, kritisch-kreative Auseinandersetzung mit digitalen Technologien und ihren kulturellen Einschreibungen. Entgegen einer rein funktionalistischen und instrumentellen Betrachtung von KI in der Programmierung geht es dabei um eine grundlegende Abschätzung von Grenzen und Reichweiten. Die Fokussierung auf Unbestimmtheit, algorithmische Artikulationsformen und die Leerstellen, die sich aus der Interaktion von Mensch und Maschine ergeben, sollten insbesondere vor dem Hintergrund erziehungswissenschaftlicher Auseinandersetzungen nicht übersehen werden, denn obwohl die Fokussierung auf Problemlösungskompetenzen heutzutage besonders wichtig sind, gelten Fehler als Normabweichungen und defizitär, doch gerade hier liegt eine direkte Verbindung von Entwicklung und Bildung. Fehler lösen Irritationen aus, so wie die Motte im Rechner, zugleich ermöglichen sie jedoch erst Reflexionsmomente und brechen durch die Anerkennung des Unerwarteten auch eingefahrene Routinen auf.

II. DAS KONNEKTIONISTISCHE PARADIGMA

SPEC ACTS. FORMALES LESEN IN REKURRENTEN NEURONALEN NETZEN

Matthew Kirschenbaum

In Wirklichkeit spricht die Maschine zur Maschine, bevor sie zum Menschen spricht, und die ontologischen Gebiete, die sie aufdeckt und absondert, sind in jedem Fall singular und prekär.

– Félix Guattari¹

I.

Am 25. März 2017 um 9:17 Uhr morgens setzte sich Ross Goodwin ans Steuer seines Schreibgeräts und begann, seinen Roman zu fahren.²

Der Roman heißt *I the Road*. Das Schreibgerät ist eine gemietete schwarze Cadillac-Limousine, die mit einer Digitaluhr, einem GPS-Ortungsggerät auf dem Dach, einem Innenmikrofon und einer beweglichen, zoombaren Außenkamera ausgestattet ist, die so eingestellt wurde, dass sie alle 20 Sekunden ein Bild schießt. Auf dem Rücksitz befindet sich ein Laptop, auf dem Machine-Learning-Modelle laufen – sogenannte neuronale Netze – und der mit einem Thermopapierdrucker verbunden ist. Während Goodwin den Cadillac steuert, registriert die Uhr

Dieser Text erschien zuerst als „Spec Acts. Reading Form in Recurrent Neural Networks“ in: *ELH* 88:2 (2021), S. 361–386.

¹ Félix Guattari, „Die maschinische Heterogenese“, in: ders., *Chaomose*, Wien: Turia und Kant 2014, S. 47–76, hier S. 65.

² Ich danke den Organisator:innen und Teilnehmenden des Jahrestreffens 2019 am English Institute der University of Chicago für ihre Aufmerksamkeit, für ihre Fragen und anregenden Gesprächsbeiträge. Ich danke David Kurnick, Paul Saint-Amour und Ted Underwood für ihre Kommentare während des Redigats meines Beitrags für die Veröffentlichung. Widmen möchte ich diesen Aufsatz Don Byrd. Er war es, der mich auf diese Richtung gebracht hat.

die Zeit sekundengenau; das Mikrofon zeichnet Gespräche zwischen ihm und seinen Fahrgästen auf; das GPS übermittelt seine aktuellen Koordinaten an den sozialen Ortungsdienst Foursquare; die Kamerabilder werden von einem der neuronalen Netze algorithmisch gesichtet und beschriftet. Anschließend werden all diese zeitlichen, stimmlichen, lokalisierten und visuellen Eingaben durch ein zweites neuronales Netz gejagt, das anhand eines Korpus bereits existierender Texte trainiert wurde, die von Genreliteratur wie Science-Fiction über Lyrik bis hin zu Werken reichen, die Goodwin einmal schlicht als „düster“ bezeichnete. Dieses zweite neuronale Netz ist für den finalen Output des Fahrgeräts verantwortlich, der auf Kassenzettpapier ausgedruckt wird – überquellende Papierrollen füllen den Rücksitz.

Begleitet wurde Goodwin von seiner Schwester, seiner Verlobten und einem Ingenieur aus der Artists and Machine Intelligence Group von Google, dazu von einem Kamerateam, das ihm in einem Kleinbus folgte. In vier Tagen lenkte er sein, wie er es nannte, „WordCar“ rund 1.300 Meilen von Bushwick, Brooklyn, nach New Orleans. Wenn man ihn ausspricht, dann fügt der Titel *I the Road* seinem offensichtlichen Vorgänger, dem Roman *On the Road* von Jack Kerouac, einen Buchstaben hinzu. Ein Jahr darauf brachte der französische Verlag Jean Boîte Éditions eine Auswahl der Papierrollen (die selbst eine Hommage an Kerouacs Schreibmaschinenrollen darstellen) in einem attraktiven Band heraus. Der Verlagskatalog bewirbt das Buch (vielleicht etwas zu selbstsicher) als „den ersten maschinengeschriebenen Roman“. Goodwin selbst würde sich nicht als Autor des Buches bezeichnen, sondern als „writer of writer“, als Autor des Autors, der für ihn aus einer Assemblage von Sensoren, Software, Silikon, Daten und Stahl besteht. Der gedruckte Text umfasst 100 Seiten, die jeweils etwa ein halbes Dutzend der mit Datum und Zeitstempel versehenen, maschinell erstellten, in Tweet-Länge gehaltenen Äußerungen der

Apparatur beinhalten. Ich werde sie von nun an als *Ticks* bezeichnen, wie das Ticken eines Kilometerzählers. Der Tick, mit dem das Buch beginnt, lautet: „Es war neun Uhr siebzehn am Morgen, und das Haus war schwer.“³

I the Road ist ein exemplarisches Beispiel für das, was demnächst die weltweit häufigste Klasse von Texten ausmachen wird; es ist ein Nachfahren dessen, was Félix Guattari einmal als jene Maschine vorhergesehen hatte, die, „zur Maschine spricht, bevor sie zum Menschen spricht“,⁴ ausgeführt als eine Anwendung im weiten technologischen Feld des *natural language processing*. Ein stetig wachsender Teil dessen, was wir täglich lesen, wird von Algorithmen, Computern und künstlicher Intelligenz geschrieben. Besonders im Journalismus ist dieser Wandel eklatant: Bestimmte Genres, die von Natur aus datenabhängig sind (zum Beispiel die Wettervorhersage, die Sport- oder Finanzberichterstattung), können heute äußerst effizient von textgenerativen Algorithmen verfasst oder mitverfasst werden. Anwendungen in anderen Bereichen – juristische Schriftsätze, Arztbriefe oder auch akademische Arbeiten – werden folgen. Der Standard solcher Fähigkeiten wurde mit der Veröffentlichung eines Sprachmodells namens GPT-3 Anfang 2020 massiv angehoben. Das von OpenAI, einem Start-up aus der kalifornischen Bay Area, entwickelte GPT-3 verfügt über 175 Milliarden maschinelle Lernparameter (eine exponentielle Steigerung gegenüber seinen Vorgängern) und produziert Texte, die in einer Vielzahl von Inhaltsbereichen als menschliche Prosa durchgehen.⁵ Zu GPT-3s demonstrativen Leistungen gehört ein

³ [Autor des Autors] Ross Goodwin, *I the Road*, Paris: Jean Boîte Éditions 2018, S. 41. Zitate aus diesem Text werden im Folgenden durch die Angabe der Seitenzahl in Klammern markiert. *I the Road* wurde bei seinem Erscheinen eine bescheidene Aufmerksamkeit zuteil. Rezensionen oder Erwähnungen gab es Ende 2018 u.a. in *The Atlantic* und *Bomb Magazine*.

⁴ Guattari, „Die maschinische Heterogenese“, S. 65.

⁵ Nachrichten und Informationen zu GPT-3 finden sich im OpenAI-Blog: <https://openai.com/blog/openai-api/> [Letzter Zugriff: 17.7.2023].

Essay, der unter seinem Namen mit dem Titel „Ein Roboter hat diesen Artikel geschrieben. Hast du schon Angst, Mensch?“⁶ im *Guardian* erschien. Solche Aktionen haben ein breites Medienecho und etliche Kommentare über die Unantastbarkeit menschlicher Autor:innenschaft hervorgerufen.

Nicht alle Technologien natürlicher Sprachverarbeitung beruhen auf neuronalen Netzen des von Goodwin verwendeten Typs. Doch ihnen allen sind einige bekannte Grundsätze dessen gemein, was man „Maschinelles Lernen“ oder „Deep Learning“ nennt. Im Kern geht es darum, dass ein Computersystem (oder Netzwerk oder Modell) anhand eines bereits vorhandenen Korpus darauf trainiert wird, Prosa zu produzieren, die ihren Vorgängern ähnelt.

Bei ihrer Implementierung verlassen sich solche Systeme auf Vorhersagemechanismen, die sich kaum von der Art und Weise unterscheiden, wie Ihr Telefon Ihnen Vorschläge für das nächste Zeichen macht, das Sie tippen könnten, für das nächste Wort, das sich zur Verwendung anbieten würde, oder sogar für den nächsten Satz, den zu schreiben Sinn ergäbe. Solche Funktionen sind statistisch gesteuert. Man kann sagen, dass sie in dem Sinne ‚lernen‘, als dass sie stetig besser darin werden, jenen Mustern zu entsprechen, welche die Maschine auf einer Makroebene wahrnimmt und die für die menschliche Wahrnehmung zu unterschwellig sind. Mit jeder Iteration erweitert und präzisiert der Algorithmus seine Mustererkennung.

Dieser Text ist kein Versuch, eine Genealogie oder Poetik des maschinellen Schreibens zu entwerfen – dieses Projekt haben andere schon in Angriff genommen. Er ist auch nicht als Polemik gegen oder als Lobrede auf das Phänomen gedacht. Stattdessen sollen zwei Behauptungen über eine bestimmte technologische Implementierung – über das in meinem Titel genannte

⁶ Vgl. GPT-3, „A robot wrote this entire article. Are you scared yet, human?“, in: *The Guardian*, September 2020, <https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3> [Letzter Zugriff: 17.7.2023].



Abb. 1:
GPS-Sensor auf dem Dach des WordCar.
Die Abbildungen 1 bis 3 stammen aus dem Film
AUTOMATIC FOR THE ROAD von Lewis Rapkin (2018).

Abb. 2:
Die Unterseite der Überwachungskamera
mit der Aufschrift „further“ (AUTOMATIC FOR THE ROAD)

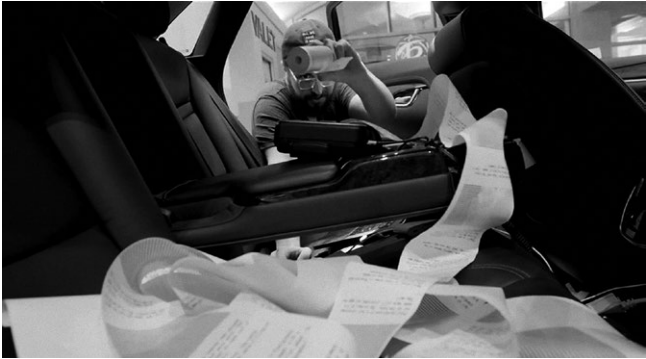


Abb. 3:
Goodwin sammelt ausgegebene Thermopapierrollen (AUTOMATIC FOR THE ROAD)

recurrent neural network – und ihre Auswirkungen auf die Literaturwissenschaft bzw. auf die Literaturgeschichte aufgestellt werden. Die erste Behauptung ist, dass rekurrente neuronale Netze (deren Funktionsweise ich unten genauer erklären werde) Agenten oder Entitäten sind, die der reinen Form so nahe kommen, wie es uns wahrscheinlich sonst nirgendwo begegnet. Sie widerstreben und widerstehen unseren gewöhnlichen materialistischen und sozialkonstruktivistischen Interpretationsansätzen und fordern uns zu anderen Lesestrategien auf. Neuronale Netze sind formale Entitäten, die die Welt auf formalistische Weise lesen und dabei Lesarten produzieren, die wir indirekt und partiell durch die Beschäftigung mit dem verstehen können, was ich ihm Folgenden einen Vektorraum nennen werde. Auf diese Weise – so werde ich argumentieren – treten neuronale Netzwerke auch in ein bestimmtes Verhältnis zu den Grundsätzen und Annahmen verschiedener Formalismen, von denen die heutige Literaturwissenschaft geprägt ist.

Meine zweite Behauptung folgt aus der ersten. Als rein formale Gebilde fordern neuronale Netze auch das Gebot der His-

torisierung heraus, das unser Studium aller wichtiger literarischer Gattungen und Epochen dominiert hat. Joseph North war derjenige, der diese pauschale Ehrerbietung gegenüber den Spezifika der Vergangenheit am heftigsten attackiert hat. Er erkennt in ihr einen tief verwurzelten Neoliberalismus, ja ein verkapptes politisches Programm, das jede praktische Kritik und Ästhetik verdrängt und das radikale Potenzial der Literaturwissenschaft neutralisiert.⁷ Neuronale Netze sind textuelle Agenten, die dem historisierenden Denken absolut zuwiderlaufen. Sie sind progressiv und unidirektional. Ein neuronales Netz weiß wenig über *agency* oder Kausalität, es beruht ganz allein auf der Kenntnis seiner eigenen gegenwärtigen (und typischerweise vorübergehenden) Konfiguration und Gewichtung. Das soll nicht heißen, dass neuronale Netze selbst nicht historisiert werden können: Der Begriff geht eindeutig auf einen Aufsatz von Warren S. McCulloch und Walter Pitts aus dem Jahr 1943 zurück, zwei Autoren, deren Namen stark mit der im darauffolgenden Jahrzehnt bekannt gewordenen Kybernetik verbunden sind.⁸ (Kybernetik: vom griechischen *kubernētēs*: Steuermann oder Fahrer, Pinne oder Rad.) Die eigentliche Epistemologie des maschinellen Lernens – wie ein neuronales Netz lernt und etwas über die Welt erfährt – ist entschieden antikausal. Stattdessen ist sie gnadenlos und ausschließlich iterativ, immer nur nach vorne gerichtet. Diese Orientierung spiegelt sich auch in den *behavioral futures*-Märkten wider, über die Shoshana Zuboff geschrieben hat und die eine kommerzielle Anwendungen der Machine-Learning-Technologie sind. Insofern, als die prädiktiven Funktionen der Technologie und die prädiktiven Imperative des Marktes übereinstimmen (ein Thema, auf das ich im letzten Abschnitt zurückkommen

⁷ Vgl. Joseph North, *Literary Criticism. A Concise Political History*, Cambridge, MA: Harvard University Press 2017.

⁸ Vgl. Warren S. McCulloch und Walter Pitts, „A Logical Calculus of the Ideas Immanent in Nervous Activity“, in: *Bulletin of Mathematical Biophysics* 5 (1943), S. 115–133.

werde), soll dieser Aufsatz auch einer Verschiebung oder Transformation Rechnung tragen, die ganz unabhängig von der Literaturwissenschaft dazu geführt hat, dass der vorherrschende epistemologische Rahmen der westlichen Ordnung nicht mehr streng historistisch ist.

Als Antonym oder Gegenmittel zu allen Historizismen bietet sich das Spekulative an – eine Kategorie, die in zahlreichen Diskursen und Domänen, von der Finanzwelt über die Philosophie bis hin zur Belletristik, bereits voll zum Einsatz kommt.⁹ „Spekulation ist unser Zeitgeist“, heißt es in einem kürzlich veröffentlichten Schreiben eines anonymen Autor:innenkollektivs, das sich selbst *Uncertain Commons* nennt. „Wir leben in einer Welt, die von Spekulationen geprägt ist“, heißt es weiter. „Mehr und mehr, so scheint es, wird die Zukunft in die Gegenwart importiert, sie wird gebündelt, verkauft, instrumentalisiert. [...] Im Grunde ist das nichts anderes als Spekulation.“¹⁰ Dies ist eine verblüffende Diagnose, und meine Aufgabe hier gestattet mir nicht, ihr in alle ihre geheimnisvollen Verästelungen zu folgen: in die Risikoanalyse, die prädiktive Genomik, die Finanzarbitrage, die *behavioral futures* und so weiter. Das Wesentliche daran ist der nahende *future collapse*, ein Zusammenbruch der Zukunft. Die Fülle des Zukünftigen wird von all diesen postdisziplinären Wissenschaften, die man unter dem Begriff der Spekulation zusammenfassen kann, gebündelt, kommerzialisiert und in handlungsleitende Outputs verwandelt. Dies sind die ‚Spec Acts‘ oder ‚Spekulationsakte‘, auf

⁹ Vgl., neben einer Reihe neuerer Arbeiten, Aimee Bahng, *Migrant Futures: Decolonizing Speculation in Financial Times*, Durham: Duke University Press 2017. Für die spekulativ-realistische Bewegung in der Philosophie bildet Levi Bryant, Nick Srnicek und Graham Harman (Hg.), *The Speculative Turn: Continental Materialism and Realism*, Victoria: re.press 2011, gewöhnlich den Ausgangspunkt. Johanna Druckers *SpecLab. Digital Aesthetics and Projects in Speculative Computing* (Chicago: University of Chicago Press 2009), bietet einen Ausblick auf die *digital humanities*. Zu den Autor:innen, die die Spekulation (in stets unterschiedlichen Ausprägungen) hervorgehoben haben, gehören auch Lisa Lowe und Saidiya Hartman.

¹⁰ *Uncertain Commons, Speculate This!*, Durham: Duke University Press 2013, S. 40.

die ich in meinem Titel anspiele. Ich habe diesen Titel bewusst in Anlehnung an J. L. Austins und John Searles Begriff der illokutionären Sprechakte, der ‚Speech Acts‘, gewählt.

Indem ich mich auf den Sprechakt beziehe, möchte ich dem intrinsischen Humanismus von Schrift und Sprache entgegen treten – es geht mir hier nicht um menschliche Sprechakte, sondern um Spekulationsakte, das heißt um algorithmische Ereignisse, die von Maschinen initiiert und ausgeführt werden. (Es sind keine einzelnen Ereignisse, sondern Millionen, Milliarden, jeden Tag, auf der ganzen Welt, wenn eine Iteration in die nächste greift und eine tosende Kaskade von Maschinen in Gang setzt, die zu Maschinen sprechen.) Ich möchte auch daran erinnern, dass Spekulationsakte (genau wie Sprechakte) Folgen haben: Algorithmen führen Handlungen aus, sie bestimmen zum Beispiel Ihre Kreditwürdigkeit oder die Zusammenstellung Ihrer Suchergebnisse. ‚Spec Act‘ ist meine eigene Begriffsbildung, der Idee nach folge ich aber Louise Amoore, die die definierende Eigenschaft eines Algorithmus in seiner „Fähigkeit“ sieht, „aus einer Reihe von Attributen einen verwertbaren Output zu erzeugen“.¹¹ Amoore nennt das Beispiel eines Bilderkennungs-Algorithmus, der die Daten einer Drohne auswerten soll, um festzustellen, ob es sich bei einem Fahrzeug am Boden um einen Schulbus oder ein militärisches Ziel handelt. Die Illokution des Algorithmus besteht darin, dass er einen Möglichkeitsraum überwacht und aus einer Vielzahl möglicher Zukünfte eine einzige destilliert: Er trifft die handlungsleitende Entscheidung, das Fahrzeug anzuvisieren oder nicht.¹² Der Spekulationsakt benennt diesen illokutionären Moment, in dem die Mannigfaltigkeit der Zukunft in eine verwertbare Gegenwart kollabiert, die so absolut und unerbittlich ist wie der Vektor eines Sprengkopfes.

¹¹ Louise Amoore, *Cloud Ethics. Algorithms and the Attributes of Ourselves and Others*, Durham: Duke University Press 2020, S. 4.

¹² Ebd., S. 17.

Vor drei Jahrzehnten beschrieb Don Byrd in einem vorausschauenden, aber wenig beachteten Buch, das eine inzwischen bekannte intellektuelle Genealogie nachzeichnet, wie die Verflechtung von Naturwissenschaften, Mathematik und Philosophie eine neue Epistemologie hervorgebracht hat. Byrd bringt es auf eine einfache, aber einschneidende Formulierung: „Anstatt die Welt als eine Maschine zu begreifen, die durch einen logischen Formalismus beschrieben wird, beschreibt der logische Formalismus alle möglichen maschinenähnlichen Welten.“¹³ Einige Seiten später kehrt er Jean Baudrillards berühmte Definition des Realen als „das, wofür es möglich ist, eine äquivalente Repräsentation herzustellen“ um: „Eine mögliche Definition des Realen“, so Byrd, sei in Wahrheit das, wofür es „*unmöglich* ist, eine äquivalente Repräsentation herzustellen“.¹⁴ Jedes ‚Bit‘ an Information in unserer formalisierten Welt ist nur zu sich selbst äquivalent, es ist ein diskretes Token, dessen Mangel an Abhängigkeiten es für die Manipulation in Systemen verfügbar macht, die durch ihre augenblicklichen Konfiguration definiert ist – relational, kontingent und vor allem statistisch singulär und diskret. „Dies ist die Realität, die sich niemals wiederholt“, schließt Byrd. „Ihre Technologie produziert nichts als ihre eigene Identität, deren Konstanz darin liegt, sich stets wechselnd selbst zu realisieren.“¹⁵ Er könnte hier einen buchstäblichen Bit gemeint haben, eine dieser Cartoon-Einsen und Nullen, die angeblich in unseren Computern herumschwirren. Vielleicht hatte er aber auch die in unserem Alltag allgegenwärtigen Attacken von Big Data im Sinn, also so banale Dinge wie Restaurantempfehlungen, die wir erhalten, während wir eine bestimmte Straße entlanggehen, während unser Smartphone Rechenkaskaden auslöst, die nahezu

¹³ Don Byrd, *The Poetics of the Common Knowledge*, Albany: The State University of New York Press 1994, S. 275f.

¹⁴ Ebd., S. 280.

¹⁵ Ebd.

unendliche Permutationen von Möglichkeiten hervorbringen, Algorithmen innerhalb von Algorithmen, Agenten innerhalb von Agenten, während ein stetig wachsender Wirbel von Variablen – Standort, Kompatibilität, frühere Outputs – mobilisiert wird, um einen Kandidaten für unsere nächste Portion Ramen zu bestimmen. Wenn ein Zyklus oder Szenario genauso verfügbar ist wie der nächste, dann wird die Realität zu dem, wofür es logischerweise gar keine äquivalente Repräsentation mehr geben kann – denn jedes Ereignis ist so brutal einzigartig wie der Moment eines Aufpralls.¹⁶

Im Fall von *I the Road* nimmt der von dem Kollektiv Uncertain Commons diagnostizierte Zukunftskollaps die Form eines gedruckten Romans an. (Goodwin selbst beschreibt sein Buch als „einen – leider sehr verstümmelten – Liebesbrief aus der Zukunft“.¹⁷) Mithilfe von Sensoren und Software wird das kombinatorische Potenzial von Text in konkrete Wörter und Sätze umgesetzt, Schriftzeichen für Schriftzeichen, eins folgt dem anderen gemäß einer momentanen Konfiguration statistischer Gewichtung. Alphabetische Zeichen, absolut in ihrer diskreten Identität, verschmelzen zu Wörtern, Sätzen, Geschichten – dieses und nicht jenes, ein Buchstabe nach dem anderen, hitzeversiegelt auf unbedarften Papierrollen, die schließlich (und das heißt hier: so gut wie zeitgleich) aus dem Thermodrucker

¹⁶ In diesem Zusammenhang denke ich an Manuel DeLandas Bemerkung, dass er ein „unapologetischer“ Realist wurde, nachdem er sein Buch *War in the Age of Intelligent Machines* (1991) geschrieben hatte. DeLanda verstand ein Schlachtfeld (ob beherrscht von Armbrüsten oder Marschflugkörpern) als einen Ort, „der von metallischen Geschossen, Schrapnellen, Schockwellen und Feuer erfüllt ist“, von irreduziblen unmenschlichen Absolutheiten, die verstümmelte Körper und Leichen hinterlässt, ganz egal, welchen ontologischen Status deren Besitzer:innen haben. In unserem eigenen, hoffentlich etwas prosaischeren Leben birgt jeder Weg zur Arbeit und jede Autofahrt die entfernte Möglichkeit eines solchen *reality check* mit roher Gewalt. Vgl. DeLanda und Graham Harman, *The Rise of Realism*, Cambridge, MA: Polity Press 2017, S. 3.

¹⁷ Vgl. „An AI and an artist go on the road. The idea was to write a novel with a car.“, in: *CBC Radio*, Oktober 2018, <https://www.cbc.ca/radio/spark/409-1.4860495/an-ai-and-an-artist-go-on-the-road-the-idea-was-to-write-a-novel-with-a-car-1.4860760> [Letzter Zugriff: 17.7.2023].

hervorquellen. Es dürfte kein besonders einfach weiterzuverarbeitendes Produkt gewesen sein: Man kann sich vorstellen, wie sich die langen Papierschleifen sich verheddern und verdrehen und in eine wogende, chaotische Fläche ausufern, wie ein achtlos abgespultes Tonband oder eine verschlungene Straße.

2.

I the Road stellt sich selbstbewusst in die Tradition der amerikanischen *road novel*: seine Vorbilder sind *On the Road* (natürlich), aber auch Tom Wolfes *Electric Kool-Aid Acid Test*, Hunter S. Thompsons *Hell's Angels* oder *Fear and Loathing in Las Vegas*. All das sind Werke, die Goodwin auf Nachfrage zu erwähnen pflegt. (In einer Geste, die fast zu selbstbezogen wirkt, ist das Wort „further“ auf die Unterseite der WordCar-Kamera eingebrannt, so wie es auf der Vorderseite des Busses der Merry Pranksters prangte [siehe Abbildung 2].) Bedauerlicherweise zitiert Goodwin *nicht* Ed Ruschas fotografisches Daumenkino *Royal Road Test* (1963), das liebevoll das Massaker dokumentiert, das sich am Straßenrand des High Desert Interstate südwestlich von Las Vegas entfaltete, nachdem eine Schreibmaschine aus dem Fenster eines in voller Fahrt befindlichen Buick Le Sabre geworfen worden war.

Kerouac und seine Verbündeten hatten ihren Bebo und ihren Schnaps, und die Prankster hatten LSD. Goodwin ist high auf KI. Das sagt er in einem Voice-over-Segment des Kurzfilms, der anlässlich der Reise produziert wurde, während vor dem Fenster Ausschnitte der New Jersey Turnpike vorbeiziehen:

Dies ist eine Maschine, die am Anfang ihres Lernprozesses nur ein zufälliges Durcheinander von Buchstaben, Leerzeichen und Satzzeichen ausspuckt. Nicht einmal Wörter gibt

sie aus. Es ist eine Maschine, die sich von Menschen geschriebene Bücher anschaut und daraus lernt, was Wörter sind, was Grammatik ist, wie man aus Ideen sinnhafte Sätze baut. Sie ist nicht ganz auf menschlichem Niveau. [...] Eher auf dem Niveau eines Insektenhirns, das Schreiben gelernt hat.¹⁸

Es folgt ein kurzes Interludium mit Ambient-Musik, dann fährt Goodwin, während er auf dem Rücksitz die endlosen Papierbänder des Thermodruckers liest, fort:

Wir sind in New Jersey, die Maschine ist los und schüttelt sich die Beine aus. [...] Es war wirklich interessant zu sehen, wie die Texte entstanden sind. [...] denn ich hatte es gerade noch in meiner Wohnung vorgeführt, und wenn man dann jetzt sieht, wie sich die Szenerie so schnell verändert, dann ist das wie die erste Reise eines Babys. Ich würde nicht sagen, dass es lebt, aber ich würde es als ein Wesen bezeichnen – zum ersten Mal zeigen wir ihm die Welt.

Am nächsten Morgen produziert das WordCar einen Tick, der einfach besagt: „Es war eine seltsame Sache“ (S. 66). Es ist unmöglich zu wissen, worauf sich dieser Tick bezieht oder bezog, aber es ist natürlich verlockend, ihn selbstreflexiv zu deuten: dass das WordCar hier von sich selbst, seinem eigenen Output, seiner nicht unbedingt langen, aber unbestreitbar seltsamen Reise spricht.

Die Ticks (was, wie ich schon sagte, meine eigene, selbst gefundene Bezeichnung für die Ausgaben des WordCar ist) sind kurze, impressionistische, bisweilen halluzinatorische, gelegentlich nachvollziehbare Prosa-Emissionen. Sie spulen sich auf dem Bandpapier ab wie der Seitenstreifen der Straße,

¹⁸ Zitiert nach *AUTOMATIC FOR THE ROAD*, Regie: Lewis Rapkin (2018), <https://www.youtube.com/watch?v=TqsWoPMd8Ro> [Letzter Zugriff: 17.7.2023].

der unter unseren Reifen vorbeirauscht – ein Effekt, der durch die schmale Spaltenbreite, in der die Papierrollen im Jean Boîte-Band abgebildet sind, nachgeahmt wird (siehe Abbildung 4).

Wie aus Tabelle 1 hervorgeht, *hat* dieser Text eine Struktur in Form von Syntax und wiederkehrenden Entitäten, die den Möglichkeitsraum des Buchs konturieren. Sie schaffen eine Umgebung oder Ökologie, ja eine *Ontologie* für einen fundamentalen, navigierbaren Erzählraum. Diese Ontologie gründet sich auf Objekte wie Gebäude, Häuser, Straßen und Bäume. Es gibt auch feinere Abstufungen des architektonischen Raums wie Fenster, Türen, Böden und Treppen, Scheunen und Fabriken, Straßen, Wege und Pfade, Züge und Bahnhöfe. Die natürliche Welt ist mit Gras und Wind, mit Mond, Sonne und Sternen gefüllt. Diese und andere Elemente tauchen im gesamten Buch auf und verleihen ihm eine kohärente, wenn auch nicht immer kontinuierliche innere Landschaft.

Bei der Lektüre von *I the Road* schaut man auf die gedruckten Seiten und sucht nach Anzeichen eines Ziels oder einer Intentionalität, eines Momentums, einer Richtung. Doch der Blick schweift ab, die Gedanken schweifen weiter, man blättert weiter, drückt das Gaspedal langsam durch, bis man sich mit einem Ruck wieder in die Langsamkeit des *close reading* begibt. Tatsächlich wird die sogenannte Oberflächenlektüre (*surface reading*) zu einer verlockenden Alternative. Man kann es sich als eine Art Tempomat vorstellen, der sanft von einem Tick zum nächsten cruist, anstatt sich an symptomatischen Textfetzen festzuklammern, als ob es in den vielschichtigen Innereien der Algorithmen irgendetwas auszuloten gäbe, etwas, das die Maschine wirklich sah oder sagen wollte, was sie gesehen hätte oder hätte sagen wollen, wenn sie nicht in Wirklichkeit eine Maschine wäre, wobei sie ja nicht einmal das wirklich ist, sondern nur ein rekursives Netz.

Tabelle 1. Beispiele für „Ticks“ in *1 the Road*. **Fett gedruckte Wörter** weisen auf syntaktische Muster hin. Unterstreichungen markieren wiederkehrende Entitäten. Die Zeitangabe erfolgt nach dem Muster Stunde/Minute/Sekunde auf einer 24-Stunden-Uhr, die daran anschließende Ziffer bezeichnet den Reisetag (vom 1. bis zum 4. Tag, das heißt vom 25. bis 28. März 2017).

Es war neun Uhr siebzehn am Morgen, und das Haus war schwer. 09:17:34 I

Es war sieben Minuten vor zehn Uhr morgens, **und** das war die einzige gute Sache, die passiert war. 09:53:34 I

Was ist das?, fragte der Maler. 09:54:23 I

Ein Stück grünes Gras schien sein Gesicht zu suchen, aber es war nicht viel zu sehen. Auf dem Bürgersteig war bereits ein kleines Stück Gras ausgelegt worden, **und** die Stufen der Scheune waren verschlossen. 10:36:11 I

In der Ferne erschien ein Baum, als hätte die Straße den Himmel aufgeweicht. Der Himmel ist klar **und** die Sonne braucht nicht gestartet zu werden. 12:15:23 I

Was machst du denn hier?, fragte er. 11:21:21 I

Und da ist ein a yeah es ich sage es noch mal **und** trage das Teil ich könnte bedeuten wo wir alle Muslime sind entweder **und** nix für die Welt halten. 12:21:58 I

JFK Memorial Highway Toll Plaza: eine Mautstelle in Perryville, wo im Sommer das berühmte Bündel goldener Ziegelsteine aufgestellt wurde **und** das Land mit dem Leder des gemeinsamen Raums gesättigt. 13:11:01 I

38.141697482 N, -77.50695616 W, auf 197.38845776 Fuß über dem Meeresspiegel bei 20.431921222 Meilen pro Stunde und die vier Ecken des Ladens waren ein seltsamer Sommertag. 16:45:52 I

95 South Spotsylvania: ein bewegliches Ziel in Leawells Schwarzwald. Es war ein ziemlich schmerzhaftes Los, **und**

zum dritten Mal sahen wir, wie der Maler in das untere Stockwerk des Hauses gekurbelt wurde. 16:54:11 1

Der Tisch ist braun **und** Esau stand im Flur **und** schaute auf den Boden, der im zweiten Stock gewesen war. Teil eines Bettes von einigen Beschwerden, dass er erwartet ihn hatte, war eine zweite **und** zwanzig oder zwanzig Meilen entfernt. 20:44:48 1

Der Himmel ist dunkel **und** der Himmel ist fast grün gestreift **und** der Himmel steht fast bis zur Oberfläche. Die Sonne ist mit schwarzer Sonne gestreift. Ein Schatten im Boden fiel auf den Boden. Das Gespräch war ungehörig, als die Tür sich öffnete **und** der große Knopf noch immer still war. 20:48:20 1

Der Boden ist aus Holz, sagt der Zug, **und** die Straßen erheben sich von den Straßen **und** die Tiefen des Fensters wachsen noch. Ein Teil eines Tisches in der vorderen Halle wurde geöffnet **und** die Luft war unten **und** der Stein war offen. 22:00:59 1

Es war sieben Minuten nach zehn Uhr abends. Die Station war menschenleer. Der Weg lag bereits in der Sonne. 22:07:09 1

Ein weißer Strich an der Seite des Gebäudes war warm **und** klein, **und** ein Stockwerk war offen, als der Wind aufwehte, **und** die Kirchtürme waren blau **und** strohblaus. Ein kleines Gebäude erschien mit dem hohlen Fass in den Fenstern des niedrigen Kellerraums, **und** die Sterne waren noch da. 08:55:31 2

Ein Auto auf der Straße war klar **und** poliert, **und** im Schatten des Wagens bildete sich ein offener Weg, **und** der Mond war zu tief. Ein großes weißes Gebäude war warm **und** gerade, **und** der Boden war noch weich **und** gerade. Der Zug hielt an, **und** der Mond ging immer noch weiter. 08:59:56 2

Es war neun Minuten nach neun Uhr morgens. Die Wege waren noch fertig. 09:09.22 2

Wer *1 the Road* liest, wird schnell feststellen, dass bestimmte formelhafte Konstruktionen vorliegen, buchstäbliche Tics, die auf die ursprünglichen Eingaben der neuronalen Netze als Ausgangspunkt für jede Textausgabe zurückgeführt werden können. Das erste dieser Konstrukte, das wir im ersten Satz des Buchs sehen, ist temporal und leitet sich von der inneren Uhr des Autos her. Manchmal wird ein zeitlicher Tick mit der Vergangenheitsform eingeleitet: *The time was ...*, „Es war ... Uhr“. Beachtenswert ist auch die häufige Verwendung der Konjunktion „und“, die das neuronale Netz als alltägliches Bindeglied zwischen verschiedenen Satzelemente erkennt. Auch die GPS-gesteuerten Ausgaben folgen zwei Grundformen: Manchmal wird ein bestimmter Ort genannt: JFK Memorial Highway Toll Plaza, Patapsco State Park, I-95 Ausfahrt 163, Potomac Hills, Da Vie Nail Salon, das Waffle House, BP, das Hard Rock Café und so weiter – das GPS greift auf die bereits erwähnte Foursquare-API zu und verknüpft Standortdaten mit den von diesem Dienst katalogisierten Orten. In anderen Fällen erhalten wir die GPS-Koordinaten selbst, die in Dezimalgradschreibweise zusammen mit Daten wie Höhe über dem Meeresspiegel und Geschwindigkeit in Meilen pro Stunde dargestellt werden. Gesprächseingaben über das Mikrofon in der Fahrzeugkabine sind die am seltensten wiedererkennbaren Eingaben in diesem Buch, man kann sie aber an der Verschlechterung der grammatikalischen Qualität erkennen, die ihnen einen Anschein von Umgangssprache, von Dialekt oder Argot verleiht. Schließlich ist da noch eine Reihe einfacher deklarativer Aussagen, die von den oben katalogisierten Einheiten abweicht: „Ein Auto auf der Straße.“ „Eine weiße Linie.“ „Der Boden.“ „Der Himmel.“ Diese im Buch am häufigsten vorkommenden Aussagen gehen auf die Interpretation von Bildern zurück, die die Kamera aufgenommen hat. (Die Buchausgabe enthält auch textualisierte Reproduktionen ausgewählter Bilder, die in einer

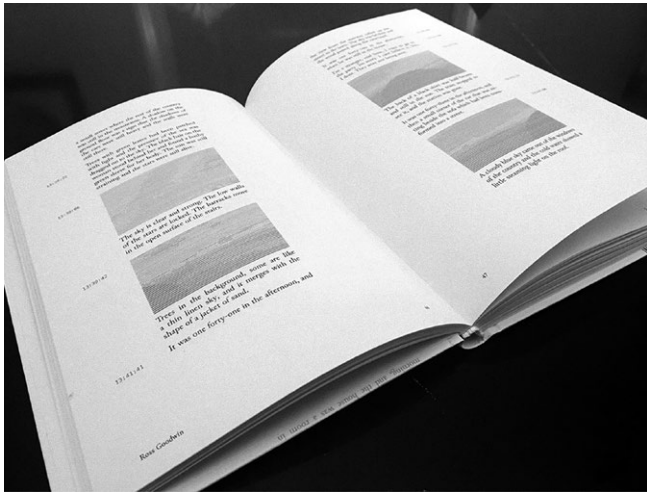


Abb. 4:
Eine Doppelseite aus der Jean Boite-Ausgabe von *I the Road*. Foto des Autors

Mikroschrift als Zeichenketten in der Art traditioneller ASCII-Kunst dargestellt werden.)

Auf den ersten Blick scheint Goodwins Experiment mehr oder weniger jener Art von Projekten zu ähneln, die Kenneth Goldsmith als „unkreatives Schreiben“ bezeichnet. Konzeptuelles (oder unkreatives) Schreiben ist eine bewusst performative Patchwriting-Praxis, die von der Überzeugung getragen ist, dass die Welt im Zeitalter des digitalen Überflusses schon mehr als genug Wörter enthält und das die Einführung neuer, eigener und originaler Sequenzen das Letzte ist, was Schreibende heute tun brauchen.¹⁹ Beispiele dafür sind solche Arbeiten wie *Day*, in der Goldsmith den kompletten Text einer Tagesausgabe

¹⁹ Vgl. Kenneth Goldsmith, *Uncreative Writing. Sprachmanagement im digitalen Zeitalter*, Berlin: Matthes & Seitz Berlin 2017.

der *New York Times* abgetippt und als Buch veröffentlicht hat, oder *Traffic*, das ausschließlich aus den transkribierten Funkmitteilungen eines 1010 WINS-Verkehrshubschraubers besteht, der zur Hauptverkehrszeit über den Brücken und Arterien Manhattans schwebt. Doch trotz all seiner Mechanisierung ist das unkreative Schreiben auch ein zutiefst körperlicher Akt; im Fall von Goldsmith erfordert es Transkriptionsleistungen, die ohne eine immense körperliche und geistige Ausdauer nicht zu bewältigen sind. Goodwins Roman erfordert wohl nur wenig von beidem: Der „Autor des Autors“ muss weder schreiben – und damit meine ich, um es mit Truman Capote zu sagen, tippen – noch komponieren, ja nicht einmal wirklich hart arbeiten. Er muss einfach nur sein Auto fahren, *hup hup*. Vor allem aber zeichnet sich der Konzeptualismus durch seinen konservatistischen Ansatz aus, durch eine Kargheit, die darin besteht, dass kein neuer oder origineller Text produziert werden soll oder kann. Goodwins neuronale Netze hingegen tun genau das: Sie nutzen die statistischen Möglichkeiten der Sprache, um Zeilen und Sätze zu generieren, die, obwohl sie aus einem Korpus oder Kanon hergeleitet sind, so noch nicht existierten und wahrscheinlich auch nie existieren würden. In dieser Hinsicht sind sie den Methoden des Oulipo näher als Goldsmiths Konzeptualismus.

Goodwin fährt also weiter, der Motor seines Cadillacs verbraucht fossile Brennstoffe, der Kilometerzähler tickt, die Sensoren nehmen ihre Umgebung auf, die neuronalen Netze summieren, der Thermodrucker rattert. Manchmal tauchen kleine erzählerische Einschübe oder Abzweigungen auf, die fast immer der Durchquerung eines bestimmten Ortes geschuldet sind – Lorton Town Center in Nordvirginia oder Goldsboro, North Carolina, oder Auburn, Alabama oder Biloxi, Mississippi, schließlich New Orleans. Die Erfahrung des Buches ist kumulativ, ein Prozess des Eintauchens und der Akklimatisierung, aber auch eine allmähliche Absenkung der Erwartungs-

haltung, wenn Muster und Entitäten wieder und wieder vorbeiziehen, wobei sich die Möglichkeitsräume eher verengen als erweitern, selbst wenn jede einzelne Äußerung oder jeder einzelne Tick auch weiter in der Lage ist, uns zu überraschen, sogar ganz am Ende noch, wenn wir uns, wie bei jeder *road novel*, fragen, wie die nächste Reise wohl aussehen wird, wenn wir ankommen beziehungsweise lesen: „Eine weiße Linie auf dem Boden stand offen, und die Jungen standen in der Tür“ (S. 140). So lautet die letzte, wohl selbstreflexive Zeile des gedruckten Buches. Darin liegt natürlich genau der Reiz der offenen Straße für einen Kerouac oder einen Kesey – sie ist der moderne amerikanische Ort der Initiation und Entdeckung; die Freiheit der weißen Linie auf den Interstate Highways nimmt die Freiheit der Datenpakete vorweg, die sich schon bald so mühelos (jedenfalls scheint es so) durch das Internet bewegen werden, modelliert von der nächsten Generation von Ingenieuren nach dem Vorbild des National Highway Systems, von Ingenieuren, die wir uns nach den hergebrachten Erzählungen als groß gewordene Jungs vorstellen müssen, die ihre Jugend damit verbracht haben, in Garagen an Sachen herumzuschrauben, an Motoren zuerst, dann an Motherboards.²⁰

Aber es wäre falsch, zwischen den sich abspulenden Ticks zu viele Verbindungen aufspüren zu wollen. Nicht einmal wenn sie in kuratierter Form auf den Seiten des Buchs erscheinen, sollte man dies tun. Es handelt sich bei den Outputs – anders als bei den Trümmern der royalen Schreibmaschine auf dem Wüstenhighway – nicht um romantische Fragmente, und auch die Beschwörungstricks des New Historicism, die uns glauben

²⁰ Dies ist natürlich eine bewusst nostalgische und tendenziöse sowie gegenderte Erzählung. Bessere Erzählungen über die Ursprünge der Computerindustrie finden sich in Nathan L. Ensmenger, *The Computer Boys Take Over. Computers, Programmers, and the Politics of Technical Expertise*, Cambridge, MA: MIT Press 2010; Janet Abbate, *Recoding Gender. Women's Changing Participation in Computing*, Cambridge, MA: MIT Press 2012; sowie Mar Hicks, *Programmed Inequality. How Britain Discarded Women Technologists and Lost Its Edge in Computing*, Cambridge, MA: MIT Press 2017, um nur einige zu nennen.

machen, Fragmente und versteckte Figuren ließen uns Kontakt zu den Toten aufnehmen, laufen ins Leere. Es handelt sich um eine Prozessierung, in Echtzeit und in brutaler Reduktion auf das rekombinante Absolute, das den Boden der statistischen Matrix bildet, die die Ausgabe von Goodwins Thermodrucker steuert. Die neuronalen Netze erweisen sich als exzellent im Lesen von Orts- und Zeitdaten, oft unvernünftig gut im Beschreiben dessen, was sie sehen, aber schlecht im Erkennen dessen, was wirklich da ist. „Das schwarze Haar auf den Augen der Frau war noch steif, und ihr Haar war schmutzig, und ihr Gesicht war noch schwer. Sie stand auf und sagte: Du musst da sein“ (S. 130). Momente wie dieser, aufgeladen mit Potemkin'scher Gravitas, lassen uns weiterlesen. Aber sie sind schon alles, was wir bekommen werden; wir werden diese Frau nie wiedersehen, nie ihre Geschichte erfahren, nie wirklich wissen, warum wir wo sein müssen. Wir stehen wie Professor Moriarty in einem Strom von Ereignissen, sitzen benommen von Geschwindigkeit hinterm Steuer und alles, was wir sehen, ist Form, kein Grund.

3.

Form, daran erinnert uns Jonathan Kramnick, ist etwas, bei dem heute viele Disziplinen ein Wörtchen mitzureden haben.²¹ Die jüngste Beschäftigung mit Form in unserer eigenen Disziplin (ich meine den sogenannte Neuen Formalismus der beiden vergangenen Jahrzehnte) wurde von Kramnick und Anahid Nersessian nützlich zusammengefasst.²² Während Varianten der Oberflächenlektüre oder andere aufmerksamkeitszentrierte Ansätze, wie ich schon angedeutet habe, eine gewisse Grund-

²¹ Vgl. Jonathan Kramnick, *Paper Minds. Literature and the Ecology of Consciousness*, Chicago: University of Chicago Press 2018.

²² Vgl. Jonathan Kramnick und Anahid Nersessian, „Form and Explanation“, in: *Critical Inquiry* 43 (2017), S. 650–669.

lage für die Lektüre von *I the Road* anbieten können, werde ich mich hier hauptsächlich mit dem beschäftigen, was Kramnick und Nersessian als „reduktionistischen“ Formalismus bezeichnen, wie er in den Schriften von Caroline Levine und Sandra Macpherson zum Ausdruck kommt.²³ Form ist nach dieser Sichtweise überall, sei es in dem, was Kramnick und Nersessian mit Blick auf Levine als „flache“ Ontologie bezeichnen (die Form ist gewissermaßen der Grund), oder sei es, weil sie, wie Macpherson es sieht, die Materie formt – Form ist überall, vom Hohem zum Niedrigen, ob wir nun von einem Auto oder einem Gedicht, einer Brücke oder einem See, einer App auf unserem iPhone oder einem Werk öffentlicher Kunst sprechen.²⁴ Mehr als um eine Frage der Anwendungsbreite des Formbegriffs geht es in dieser reduktionistischen Sichtweise und bei dieser Art von Formalismus letztlich darum zu beschreiben, welches Verhältnis Kunst (oder Politik oder Technik oder Recht) zu jener Welt eingehen, die von Form gefüllt wird.

Am wichtigsten für meine Zwecke hier ist, dass Macpherson Form als Resultat eines Prozesses betrachtet, dass also etwas, *irgendetwas*, der Form vorausgehen muss. Dieses Etwas ist die Materie. Form, so betont sie, ist „nicht mehr – und nicht weniger – als die Form, die die Materie (sei es ein Gedicht oder ein Baum) annimmt“.²⁵ Das hier gebrauchte Verb „annimmt“ teilt uns mit, dass ein Prozess im Spiel gewesen sein muss, wenn nicht streng zeitlich, so doch zumindest transaktional. Wie aber wird diese Transaktion vollzogen? Auf welcher Grundlage findet sie statt? Nach welchem Zeitplan? Aus welcher Energie speist sie sich? Was genau wird verbraucht und was bleibt erhalten? Ich gestehe, ich weiß es nicht. Aber wir kommen, so scheint mir, nicht umhin, den Prozessen, durch die die Form Gestalt annimmt (und Gestalt gibt), Form (*eine* Form, wenn

²³ Kramnick und Nersessian, „Form and Explanation“, S. 654.

²⁴ Ebd., S. 65.

²⁵ Sandra Macpherson, „A Little Formalism“, in: *ELH* 82:2 (2015), S. 385–405, hier S. 390.

auch nur ein bisschen) zuzusprechen. Macpherson beruft sich hier auf Roland Barthes, der schrieb: „Ein Baum ist ein Baum. Ja, gewiss.“²⁶ Aber nicht irgendein Baum: Für Barthes ist es der Baum des Dichter-Wunderkindes Minou Drouet im titelgebenden Poem ihres ersten Gedichtbandes *Arbre, mon ami* (1957). Ein so ausgedrückter Baum, so Barthes, „ist schon nicht mehr ganz ein Baum, sondern ein geschmückter Baum, einer, der für einen bestimmten Konsum zurechtgemacht wurde, der mit literarischen Gefälligkeiten, Revolten, Bildern versehen, kurz: für einen bestimmten gesellschaftlichen Gebrauch ausgestattet ist, der zu der reinen Materie hinzutritt“.²⁷ Auf die Wörter „nicht mehr ganz“ scheint es anzukommen: Es hat eine Transaktion stattgefunden, und danach spricht Barthes ganz klar von dem, was wir Form nennen würden. Der Baum, der einst *nur* ein Baum war („reine Materie“), erhält nun eine literarische, politische, soziale Dimension.

Bäume, so stellt sich heraus, sind nicht nur überall bei Barthes zu finden, wie Macpherson uns in Erinnerung ruft; sie sind auch eine der präsentesten Entitäten in *I the Road*, wo alle ein, zwei Seiten von ihnen die Rede ist. (Mehr als alles andere mag dies einfach ein Hinweis darauf sein, dass die Bilderkennung des Netzes seine Arbeit getan hat: Wenn man östlich des Mississippi durchs Land fährt, sieht man wirklich viele Bäume.) Betrachten Sie diesen Tick vom ersten Tag der Reise:

Bäume mit grünen Blättern waren mit Licht gefleckt worden, und der Druck des Meeres wurde an den Himmel gezerrt. Das schwarze Haar an der Frau stand hinter ihr und fand einen buschigen grünen Ärmel für ihren Körper. Die Sonne mühte sich noch und die Sterne waren noch lebendig. (S. 46)

²⁶ Zitiert nach ebd., S. 390.

²⁷ Roland Barthes, „Der Mythos heute“, in: ders., *Die Mythologien des Alltags*, Berlin: Suhrkamp 2010, S. 252.

Dieser Text mag schräg oder abstrakt sein, aber er ist nicht ohne Form. Ich weiß schon, was es bedeutet, wenn Bäume grüne Blätter haben und schwarzes Haar „an“ einer Frau ist, aber ich bin mir weniger sicher, was es bedeuten soll, dass dieses Haar hinter der Frau „stand“ oder dass es einen buschigen grünen Ärmel für ihren Körper „fand“. (Hat dieser grüne Ärmel irgendetwas mit den grünen Blättern zu tun?) Ungeachtet dessen, dass die grünen Blätter „mit Licht gefleckt“ sind, scheinen sie ganz so zu sein, wie sie sein sollten, und dass der „Druck des Meeres an den Himmel gezerrt [wurde]“, kann schon seine nautische und barometrische Richtigkeit haben – obwohl ich nicht wüsste, was dies für die Bäume und ihre Blätter bedeutet. Die Parallele zwischen einer Sonne, die sich noch müht, und Sternen, die noch leben, ist einfach und kühn, vielleicht sogar schön. Kurzum, unabhängig von der Schuldigkeit, die diese Äußerungen des WordCar im Hinblick auf normatives oder sogar grammatikalisches Englisch vielleicht noch haben, ähneln sie den Blankversen Drouets doch viel mehr als ungeformter Materie.

Wo aber ist in einem neuronalen Netz, das seinen Text auf die Schleifen eines Quittungspapiers zuckelt, eigentlich die ‚Materie‘? Jeder kompetente Medienwissenschaftler:in würde darauf mit einer materialistischen Attacke replizieren und sagen, dass die Materie all dem Material inhärent ist, aus dem der Laptop gemacht ist, der Kamera, dem Auto, auch dem Autobahnssystem, außerdem den Schichten sozialer und wirtschaftlicher Beziehungen, welche die Arbeit lenken, sich das resultierende Kapital aneignen, den Abfall entsorgen und die Bits abspeichern (in klimatisierten Rechenzentren, die zur Klimakatastrophe beitragen). Das stimmt schon alles, und es ist die notwendige Erinnerung daran, dass das Digitale nicht immateriell ist.²⁸ Am eigentlichen, auf der Hand liegenden Thema –

²⁸ Die Leser:innen sollten vielleicht wissen, dass ich meine Karriere weitgehend auf der These aufgebaut habe, dass alle Medien – und insbesondere die digitalen – unausweichlich materiell sind. In gewissem Sinne fordert dieser Text die Annahmen meiner

an dem, was neuronale Netze, auch im Vergleich zu anderen Rechnerstrukturen, so interessant macht – geht es aber vorbei: In einem sehr spezifischen und technischen Sinne gibt es hier weder eine Materie noch eine Hand.²⁹

Der Informatiker und Medienwissenschaftler Adrian Mackenzie beschreibt maschinelles Lernen und neuronale Netze im Wesentlichen als eine Erweiterung traditioneller tabellarischer Darstellungen von Daten (wie in den Zeilen und Dateien einer Datenbank) hin zu einer Vektorisierung oder zu einem Vektorraum, wobei jedes unabhängige Merkmal oder jede Variable im Datensatz zu einem singulären mathematischen Operator wird, zu einem Koordinatenpunkt, der sowohl eine Bewegungsrichtung als auch eine Größe besitzt. Laut Mackenzie zeigen Vektoren „verborgene, verdeckte oder innere Räume“ innerhalb von Datendarstellungen an, die mit linearen oder tabellarischen Standardtechniken nicht visualisiert werden können.³⁰ (Während Vektoren selbst in Tabellenform wiedergegeben werden können, sind neuronale Netze exzellent darin, komplexe, nicht lineare Beziehungen zwischen einzelnen Merkmalen, wie zum Beispiel Wörtern, in mehrdimensionalen Datensätzen zu entdecken). „Manchmal“, fährt Mackenzie fort, „nehmen lernende Maschinen den Vektorraum als

eigenen früheren Arbeit heraus und versucht, einen Ort zu bestimmen, an dem meine Untersuchungen an ihre Grenzen stoßen.

²⁹ Auch hier ist Louise Amoore hilfreich, weil sie Forderungen nach „Accountability“ für Algorithmen – als ob man die Blackbox aufbrechen und den darin lauernden Homunkulus entlarven könnte – als grundlegend konservativ darstellt, als ein Symptom dessen, was sie als „hartnäckiges und unlösbares ethisch-politisches Problem“ bezeichnet, das darin besteht, „Transparenz“ als einzig mögliche Form der Rechenschaftspflicht anzusehen (*Cloud Ethics*, S. 19). Algorithmen sind in der Tat (wie sie hervorhebt) immer parteiisch und oft unlesbar. Diese Parteilichkeit ist keine Grenze, die es zu umgehen gilt, sondern die notwendige Grundlage für eine jede Ethik des algorithmischen Handelns, eine Ethik, die letztlich davon abhängen muss, was der Algorithmus tut (d.h. von seinen spezifischen Handlungen). „Algorithmen“, schließt Amoore, „geben ständig Rechenschaft über sich selbst“ (S. 19).

³⁰ Adrian Mackenzie, *Machine Learners. Archaeology of a Data Practice*, Cambridge, MA: MIT Press 2017, S. 61.

einen Raum voller ständig variierender Näherungen wahr.“³¹ In ähnlicher Weise beschreibt Louise Amoore die inneren Architekturen eines Algorithmus als „Anordnungen von Näherungen, Abständen, Intensitäten und Assoziationen“.³² Betrachten wir abermals das obige Beispiel. Man braucht gar keine direkte Korrelation zwischen dem Text, den wir sehen, und seiner mathematischen Vektordarstellung anzunehmen, um zu erkennen, dass „ständig variierender Näherung“ und „Anordnungen von Näherungen, Abständen, Intensitäten und Assoziationen“ die seltsame Grammatik des geometrischen/geografischen Zusammenbruchs (die wortwörtlichen ‚Tics‘), die diese und viele andere Passagen auszeichnen, recht gut umschreibt. An anderer Stelle schlägt Mackenzie vor, dass die Vektorisierung „einen gemeinsamen Raum erzeugt, der komplexe lokale Realitäten nebeneinanderstellt und vermischt“.³³ So könnte die Lektüre der Spekulationsakte neuronaler Netze tatsächlich aussehen, zumindest als Ausgangspunkt: Der Vektorraum ist ein Substrat, in dem „schwarze Haare an einer Frau“ tatsächlich „hinter ihr stehen“ und dort einen „buschigen grünen Ärmel“ für ihren Körper „finden“ können.³⁴

Wir können es auch anders ausdrücken: Neuronale Netze sind keine Bäume. Sie sind nicht die verzweigten Bedingtheiten traditioneller Computerprogrammierung mit eingehegten If-then-else-Routinen. Neuronale Netze sind ... eben anders. Mackenzie unterstreicht (abermals) den Unterschied: „Das Ler-

³¹ Mackenzie, *Machine Learners*, S. 63.

³² Amoore, *Cloud Ethics*, S. 14.

³³ Mackenzie, *Machine Learners*, S. 73.

³⁴ Andrew Piper drückt dies etwas prosaischer aus: „Vektorraummodelle“, erklärt er, „übersetzen die syntagmatischen Dimensionen der Bedeutung eines Wortes oder Satzes – seinen lokalen Kontext – in einen paradigmatischen Rahmen, der aus einem größeren Verteilungskontext bezogen wird“ (ders., *Enumerations: Data and Literary Study*, Chicago: University of Chicago Press 2018, S. 16). Interessanterweise bestimmt Piper eine andere Stelle im Werk von Roland Barthes als Ausgangspunkt für seine eigene Diskussion des Vektorraums: Barthes' Epiphanie einer Sprache, die „außerhalb des Satzes“ existiert (S. 13).

nen beim maschinellen Lernen hat kaum kognitive oder symbolische Grundlagen. Es unterscheidet sich von der klassischen KI dadurch, dass es bestehende symbolische, kontrollierende, kommunikative und zunehmend auch bedeutungsgebende Praktiken verarbeitet [...] und nur in Form von Gewichtungen programmatisch auf sie aufbaut“.³⁵ Konkret verwendet Goodwin sogenannte rekurrente neuronale Netze mit langem Kurzzeitgedächtnis (*long short-term memory recurrent neural networks*, LSTM-RNNs). Ein typisches neuronales Netz ist, wie der Name schon sagt, eine Rechenstruktur, die einer idealisierten Darstellung der Art und Weise nachempfunden ist, wie einzelne Neuronen im menschlichen Gehirn Verbindungen oder Pfade knüpfen, um Funktionen auf höherer Ebene auszuführen. Jedem Knoten im neuronalen Computernetz wird ein Wert oder eine „Gewichtung“ (nach Mackenzie) zugewiesen, und diese Gewichtungen können dynamisch angepasst werden, um die Genauigkeit der Ergebnisse zu verbessern. Dies ist das klassische McCulloch-Pitts-Modell. Es ist, wenn man so will, ein Bottom-up-Ansatz für Künstliche Intelligenz, mit dem Versprechen, dass die Maschine oder das Netzwerk sich selbst trainiert. Rekurrente neuronale Netze nehmen ihre eigenen Ausgaben und speisen sie dann als Eingaben wieder in das Netzmodell ein, wodurch das Modell rekursiv wird (ein RNN mit langem Kurzzeitgedächtnis ist eine besondere Variante dieses Typs). Auf diese Weise erstellt Goodwin Sprachmodelle auf Zeichenebene, das heißt, er hat seinen Netzen eine Reihe von Beispieltexten gegeben und sie dann aufgefordert, die Wahrscheinlichkeitsverteilung des nächsten Zeichens in einer bestimmten Sequenz auf der Grundlage einer Folge vorheriger Zeichen zu modellieren. Das Netz ist in der Lage, diese Vorgänge viele Tausend Male zu iterieren. Bald lernt es, wie man buchstabiert. Dann lernt es, wie man Sätze strukturiert. Schon bald ist das Netzwerk auf ein sta-

³⁵ Mackenzie, *Machine Learners*, S. 27.

tistisches Modell abgestimmt, das Textzeichen um Textzeichen vorhersagen kann, und zwar so, dass es dem Ausgangsmaterial treu bleibt.³⁶ (Anhänger:innen computergestützter neuronaler Netze, darunter auch Goodwin, verwenden häufig Begriffe wie „trippy“, „unnatürlich“, „unheimlich“ und vor allem „magisch“, um ihre Wirkung zu beschreiben).

Anfang 2015 veröffentlichte ein polnischer Informatiker namens Andrej Karpathy eine Arbeit mit dem Titel *The Unreasonable Effectiveness of Recurrent Neural Networks*, die Goodwin fand und las.³⁷ Nach Goodwins eigener Auskunft war dies der Auslöser für seine Reise.³⁸ Sehr bald darauf veröffentlichte Google eine Reihe verblüffender Bilder, die das Internet elektrisierten und deren berühmtestes als „Trippy Squirrel“ bekannt wurde. Sehen Sie es sich genau an (Abbildung 5): Die Eichhörnchen haben Hundegesichter als Schnäuzchen. Was wir da sehen, ist das Ergebnis eines Prozesses, den Google nach dem Christopher-Nolan-Film von 2010 „Inceptionism“ genannt hat.³⁹ Das Unternehmen trainiert seine Algorithmen zur Bildklassifizierung gewöhnlich so, dass es riesige Mengen von Archivbildern durch neuronale Netze leitet und dabei die Einstellungen der Netze so lange optimiert, bis der Klassifizierer zuverlässig arbeitet. Inceptionism ist im Wesentlichen die Umkehrung dieses Mechanismus, durch den die Maschine sehen lernt. Wie die Google-Ingenieure beschreiben:

³⁶ In dem CBC-Interview „AI and Artist“ erläutert Goodwin diesen Prozess.

³⁷ Andrej Karpathys Aufsatz ist verfügbar unter: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/> [Letzter Zugriff: 17.7.2023].

³⁸ Vgl. Goodwin, „Narrated Reality“, <https://www.youtube.com/watch?v=9E22CWmOpsk>. Der Videomitschnitt zeigt einen Vortrag Goodwins über sein Forschungsprogramm auf einem Kongress namens „StrangeLoop“. Er bietet den besten verfügbaren Überblick über seine Biografie (er arbeitete zuvor als Redenschreiber für Barack Obama) und seine künstlerische Karriere.

³⁹ Vgl. Alexander Mordvintsev, Christopher Olah und Mike Tyka, „Inceptionism. Going Deeper into Neural Networks“, in: *Google AI Blog*, Juni 2015, <https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html> [Letzter Zugriff: 17.7.2023].

Wir füttern das Netz mit einem beliebigen Bild oder Foto und lassen es dieses Bild analysieren. Dann wählen wir einen Layer aus und fordern das Netz auf, die erkannten Merkmale zu verbessern. Jeder Layer verarbeitet Merkmale auf einer anderen Abstraktionsebene, sodass die Komplexität der von uns erzeugten Merkmale davon abhängt, welchen Layer wir zur Verbesserung auswählen. So neigen beispielsweise untere Layer dazu, Striche oder einfache ornamentähnliche Muster zu erzeugen, da sie auf grundlegende Merkmale wie Kanten und deren Ausrichtung reagieren.⁴⁰

Sie fahren fort:

Wenn wir höher gelegene Layer wählen, die anspruchsvollere Merkmale in Bildern erkennen, tauchen oft komplexe Merkmale oder sogar ganze Objekte auf. [...] Wir fragen das Netz: „Was auch immer du dort siehst, ich will mehr davon!“ So entsteht eine Rückkopplungsschleife: Wenn eine Wolke ein wenig wie ein Vogel aussieht, wird das Netz sie mehr wie einen Vogel aussehen lassen. Dies wiederum führt dazu, dass das Netzwerk den Vogel beim nächsten Durchgang noch besser erkennt und so weiter, bis ein sehr detaillierter Vogel erscheint, scheinbar aus dem Nichts.⁴¹

Dabei handelt es sich um dieselbe Art von rekurrenten neuronalen Netzen, die Goodwin für seine Sprachmodelle auf Zeichenebene verwendet. Google nannte das System, das diese Bilder produzierte, DeepDream und richtete ein öffentliches Portal ein, über das Nutzer:innen ihre eigenen Bilder hochladen konnten. Wie verschiedene Nachrichtenagenturen eifrig

⁴⁰ Mordvintsev, Olah und Tyka, „Inceptionism“.

⁴¹ Ebd.

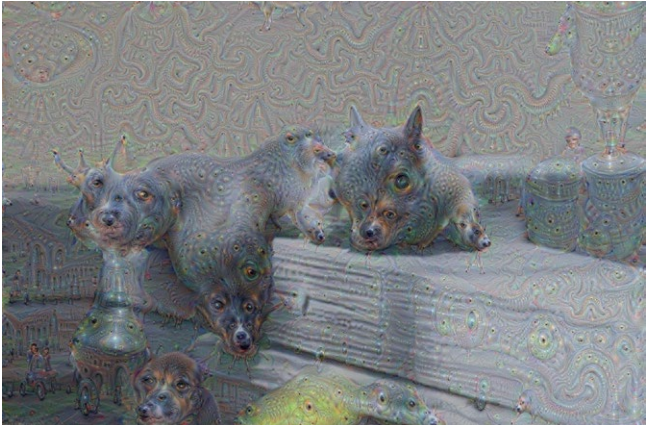


Abb. 5:
„Trippy Squirrel“ von DeepDream. Quelle: <https://i.imgur.com/6ocuQsZ.jpg>

berichteten, schienen die Ergebnisse zu zeigen, dass Androiden tatsächlich von elektrischen Schafen träumen.

Aber das Team hinter DeepDream gibt auch zu, dass „wir eigentlich erstaunlich wenig darüber wissen, warum bestimmte Modelle funktionieren und andere nicht“.⁴² In der sich entwickelnden Informatikliteratur ist dies als Reproduzierbarkeitsproblem bekannt. Praktiker:innen haben festgestellt, dass sie im Allgemeinen ihren eigenen Kolleg:innen (geschweige denn dem Rest von uns) nicht genau erklären können, wie ein einzelnes neuronales Netzwerk zu seinen Ergebnissen kommt. Shane Denson führt dasselbe Argument für ähnliche Klassen von computergenerierten Bildern an, die er als „diskorrelierte Bilder“ bezeichnet: „Die zugrundeliegenden Kontingenzen [...] befinden sich außerhalb des Bereichs subjektiver Wahrnehmbarkeit; die für den Glitch verantwortlichen Algorithmen und

⁴² Ebd.

Hardware-Operationen sind von den phänomenologischen Prozessen noetischer Intentionalität fundamental ‚diskorreliert‘.⁴³ (In den etwas plumperen Worten der Google-Ingenieure oben: Sie kommen „scheinbar aus dem Nichts“). Es ist also nicht falsch, auf der Materialität eines neuronalen Netzes zu bestehen. Aber als Erklärung reicht es nicht aus. Gewiss geben die seltenen Erden im Inneren des Laptops, die soziale Konstruiertheit der Algorithmen und die *biases* in den Datensätzen, mit denen sie trainiert werden, Anlass zu Kritik. Dennoch haben wir es hier nicht mit den kombinatorischen Maschinen der ersten Generation von Computerpoesie oder mit den prozeduralen Spielen des Oulipo zu tun, bei denen die bloße Verfügbarkeit des Mechanismus einen Teil der Werkbedeutung ausmachte.⁴⁴ Im Gegensatz zu Googles PageRank-Algorithmus, der im wahrsten Sinne des Wortes ein Algorithmus ist und als solcher in einem Paper von Larry Page und Sergey Brin veröffentlicht wurde, gibt es hier keine festgelegte Abfolge von mathematischen Operationen, die man nachbauen könnte.⁴⁵ Hier gibt es Rekursion und sonst nichts. (Die soziale Dimension solcher Phänomene – DeepDream implementiert als QAnon, als Vögel aus dem Nichts, die sich durch alle Ebenen twittern – bezeichnet Johanna Drucker als *phantasmatisch*.⁴⁶)

⁴³ Shane Denson, *Discorrelated Images*, Durham: Duke University Press 2020, S. 2.

⁴⁴ Wieder einmal finde ich Amooore sehr aufschlussreich: „Die Algorithmen des maschinellen Lernens, die ich beobachtet habe – von der Grenzkontrolle bis zur Chirurgie und von der Gesichtserkennung bis zur Betrugserkennung –, bringen Modi der Anerkennung, Bewertung und probabilistischen Entscheidungsgewichtung hervor, die zutiefst politisch sind und dennoch *nicht gänzlich auf einen erkennbaren Menschen zurückzuführen sind, der die Regeln schreibt*. Sie sind natürlich auch kalkulatorische Räume, in denen sich Vorurteile und rassistische Ungerechtigkeiten festsetzen und verstärken können, wenn auch nicht in einer Form, die mit einer Politik des ethischen Designs oder der Neufassung der Regeln ohne Weiteres zu lösen wäre.“ (*Cloud Ethics*, S. 69; Hervorhebung hinzugefügt.)

⁴⁵ Vgl. Larry Page und Sergey Brin, „The PageRank Citation Ranking: Bringing Order to the Web“, Januar 1998, <http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf> [Letzter Zugriff: 17.7.2023].

⁴⁶ Vgl. Johanna Drucker, *The General Theory of Social Relativity*, o.O., The Elephants 2018. Drucker definiert ihren Begriff des Phantasmatischen im Gegensatz zur Illusion und zum Simulakrum. Das Phantasmatische – „die affektive Auseinandersetzung mit

Was wir lesen, wenn wir neuronale Netze lesen, ist, so möchte ich argumentieren, eine Art vollständig aktivierter Formalismus, eine Form, die nicht materiell beschränkt ist und deren Manifestationen nicht länger eine notwendige Grundlage in einer vorherigen Substanz oder einem vorherigen Substrat haben. Es handelt sich um die eigentümliche Poesie eines Vektorraums. Weder der Input noch der Output sind ‚immateriell‘; aber die Transaktionen, die die Form hervorbringen, sind in jeder Hinsicht unzugänglich. Es ist nicht möglich, unter die sprichwörtliche Haube zu schauen, um den Mechanismus des Motors zu untersuchen.⁴⁷ (Die „Gewichte“, von denen die Enthusiasten des maschinellen Lernens gerne sprechen, sind schließlich nur statistische Tokens, das heißt Platzhalter für Potenzialitäten, das heißt Spec Acts.) Die Medienhistorikerin Orit Halpern fasst die Konzepte, die den „weights“ zugrunde liegen, allgemeinverständlich zusammen: „Aus dem Netz heraus kann man nicht feststellen, welche Neuronen gefeuert haben, um die aktuelle Situation zu erregen“, schreibt sie.⁴⁸ „Innerhalb eines Netzes (oder Netzwerks) sind die Grenze zwischen Wahrnehmung und Kognition, die Trennung zwischen Innerlichkeit und Äußerlichkeit und die Organisation der kausalen Zeit ununterscheidbar.“⁴⁹ Mit anderen Worten (wiederum von Halpern): „Die Zeitlichkeit des Netzes ist präemptiv,

konsensuellen Halluzinationen“ – produziert „Artefakte kollektiver Überzeugungen, die real werden, sich als Ereignisse und Bedingungen aktualisieren“ (S. 4 und 5). Ihr Beispiel ist natürlich Trump.

⁴⁷ Eine faszinierende Alternative schlagen Minh Hua und Rita Raley in ihrem Aufsatz „Playing with Unicorns: *AI Dungeon* and Citizen NLP“, in: *Digital Humanities Quarterly*, 14:4 (2020) vor. Sie untersuchen ein Textabenteuerspiel, *AI Dungeon*, das auf der GPT-2-Architektur aufbaut und den Spieler:innen iterative Korrekturen und Überarbeitungen erlaubt. Sie zeigen, wie solche Interaktionen Einblicke in die Blackbox des maschinellen Lernens geben und einen Rahmen für eine zivilverantwortliche oder ethische KI schaffen können.

⁴⁸ Orit Halpern, *Beautiful Data*, Durham: Duke University Press 2014, S. 156.

⁴⁹ Ebd., S. 157.

es operiert immer im Futur II und [...] außerhalb historischer Zeitlichkeiten.“⁵⁰

Die Implikationen sind klar: Ein neuronales Netz kennt nichts als Vorhersagen und Prognosen, und niemals die Vergangenheit. Ein neuronales Netz wird *nie* historisieren. Das Getriebe läuft ausschließlich additiv, kumulativ. Selbst die Rückkopplung (Backpropagation, in der Sprache der Zukunft) ist immer in die Zukunft gerichtet. Halpern drückt den grundlegenden Formalismus so aus: $T + I$, wobei $T = \text{Zeit}$. Wie bei den diskorrelierten Bildern von Denson gibt es kein offensichtliches Modell von Handlungsfähigkeit und Intentionalität, das uns auf die bekannten Materialismen zurückgreifen ließe. Ein Spekulationsakt besteht sowohl in diesem diskreten (oder ungezügelter) Formalismus *als auch* in der Verfügbarkeit einer epistemologischen Alternative zum Historizismus, die die mathematische Grundwahrheit – das Gegebene – dieser wahrscheinlich leistungsfähigsten jemals gebauten Maschineninstrumente ist. Oder wie Jill Lepore mit Blick auf einen anderen, früheren Kontext bemerkt: „Die Geschichtsschreibung kann die Frage ‚Was wäre wenn?‘ nicht beantworten.“⁵¹ Wie die Autor:innen der Uncertain Commons und Lepore übereinstimmend feststellen, befinden wir uns in einer Welt, die durch Spekulationsakte geschaffen wurde, von der Finanzwelt bis zu Facebook.

Form plus Form plus Form plus Form, ad infinitum. $F + I$, könnte man sagen, um Halperns Notation zu ergänzen. F steht natürlich für Form, aber auch für Further, wie es auf der Unterseite der WordCar-Kamera eingebrannt ist. Denn die Form bewegt sich in einem neuronalen Netz immer nur vorwärts –

⁵⁰ Ebd., S. 159 und S. 173.

⁵¹ Jill Lepore, *If Then. How the Simulmatics Corporation Invented the Future*, New York: Liveright 2020, S. 6. Lepores Äußerung bezieht sich auf die prädiktive (und spekulative) Datenwissenschaft, die in den 1960er-Jahren von der Simulmatics Corporation betrieben wurde und die ihrer Darstellung nach den Weg für Facebook und Cambridge Analytica bereitete.

immer weiter. Die vektorisierte Ausgabe ist immer nur iterativ; sie ist, wenn man von dem möglichen Horizont eines Wärmetods einmal absieht, grenzenlos.⁵² (Passenderweise zeichnet *I the Road* nur die Geschichte von Goodwins Hinfahrt auf: eine einfache Fahrt von Brooklyn nach New Orleans.) Macpherson, die den Historismus als Erfüllungsgehilfen der Form vehement ablehnt, schließt ihren Essay mit der Aufforderung, eine „Welt ohne uns“ als Unvermeidlichkeit anzunehmen, eine Welt, in der Formen auch dann noch *sein* werden, wenn wir längst gegangen sind.⁵³ Irgendwann muss jede Reise enden. Wenn sie dann endet, wenn die Zündung erlischt und die Verbrennungskette des Motors abbricht, kühlt das heiße Metall langsam ab und tickt dabei.

4.

Vor einigen Jahren, nachdem wir beide lange genug mit gebrauchten Karren aus dem vergangenen Jahrhundert herumgefahren waren, kauften meine Frau und ich uns endlich ein neues Auto. Sofort waren wir auf das engmaschige Netz von Sensoren und Bildschirmen angewiesen, um rückwärts in Parklücken hinein- und vorwärts wieder herauszukommen, um die Spur zu wechseln und unsere Umgebung wahrzunehmen – ein Beweis dafür, dass all die Geräte, deren Eingaben für *I for the*

⁵² Das ist keine Übertreibung. Im Dezember 2020 berichtete die *MIT Technology Review*, dass die Co-Leiterin von Googles Ethik- und KI-Gruppe ihren Job unter anderem deshalb verlor, weil ein internes Papier zeigte, dass der CO₂-Fußabdruck des Unternehmens mit der zunehmenden Nutzung großer neuronaler Netze exponentiell gestiegen war, insbesondere seit 2017. Vgl. Karen Hao, „We Read the Paper that Forced Timnit Gebru Out of Google“, in: *MIT Technology Review* (2020), <https://www.technologyreview.com/2020/12/04/1013294/google-ai-ethics-research-paper-forced-out-timnit-gebru/> [Letzter Zugriff: 17.7.2023]. – [Das „interne Papier“ wurde später veröffentlicht und ist als letzter Beitrag in diesem Band abgedruckt: Emily M. Bender u.a., „Über die Gefahren stochastischer Papageien. Können Sprachmodelle zu groß sein?“, S. 277–331. A. d. Hg.]

⁵³ Macpherson, „A Little Formalism“, S. 402.

Road an Goodwins neuronale Netze geleitet wurden, in aktuellen Automodellen längst zum Standard gehören.

In der präpandemischen Welt war eine Autofahrt statistisch gesehen die gefährlichste Aktivität, die die meisten von uns im Alltag unternehmen konnten.⁵⁴ Es ist also kein Zufall, dass Autos häufig als die ultimative Vollendung verbraucher:innen-orientierter KI angepriesen werden. Und es ist nur ein kleiner Schritt von der Innovation eines selbstschreibenden Autos zum kommerziellen Coup eines selbstfahrenden Autos. Hier reiht sich Goodwins Projekt ein in den Nexus der verdeckten Sammlung, Analyse und Ausbeutung persönlicher Daten, die Shoshana Zuboff als ‚Überwachungskapitalismus‘ bezeichnet. (Manche Passant:innen hätten das WordCar für eine Art Überwachungsfahrzeug gehalten, berichtete Goodwin – gewissermaßen stimmte das ja auch.)

Zuboff fasst zusammen: „Mit anderen Worten gehen – von Telefonen, Autos, Straßen, Häusern, Geschäften, Körpern, Gebäuden, Flughäfen und Städten aus – unablässig Aktivitäten der realen Welt zurück in die digitale Sphäre, wo sie ein neues Leben als Daten annehmen, die zur Umwandlung in Vorhersagen zur Verfügung stehen. All das füllt die Seiten des unablässig expandierenden Schattentexts.“⁵⁵ Schattentext ist der von ihr eingeführte Name für das analytische Aggregat aus Cookies, Metadaten, Serverprotokollen, Standortsignaturen und all den anderen ‚Datenabfällen‘, die ihrer Ansicht nach das Rohmaterial für Verhaltensextraktion und -vorhersage darstellen. Es sollte uns auch nicht entgehen, dass in ihrer Litanei viele der Merkmale und Objekte enthalten sind, die Goodwins Roadtrip ausmachen: Autos, Straßen, Häuser, Gebäude ... und Bäume. In dem Buch, das WordCar schreibt, wird der Schat-

⁵⁴ Man beachte, dass Radiohead 1997 ihr bahnbrechendes Album *OK Computer* mit einem Text eröffneten, in dem ein Airbag gepriesen wurde.

⁵⁵ Shoshana Zuboff, *Das Zeitalter des Überwachungskapitalismus*, Frankfurt a.M., New York: Campus 2018, S. 234.

tentext also lesbar oder zumindest sichtbar. Er ist symptomatisch für das Ausmaß, in dem, wie Zuboff sich ausdrückt, maschinelle Intelligenz der „ultimative Bandwurm“ ist und „die Intelligenz einer Maschine sich danach [richtet], wie viele Daten sie frisst“.⁵⁶ Goodwin selbst ist aufgeschlossen für diese Interpretation: „Mein nicht ganz so geheimer Hintergedanke liegt darin, den Leuten zu zeigen, wie maschinell erzeugter Text in seiner rohesten Form aussieht, damit sie für eine Zukunft, in der solche Text anspruchsvoller werden, gewarnt sind: Wenn Sie solche Muster sehen, wurde der Text möglicherweise nicht von einem Menschen geschrieben.“⁵⁷

Google ist das Unternehmen, das Zuboff – mehr noch als Facebook, Amazon oder Apple – als den größten Pionier der Technologien des Überwachungskapitalismus und auch als den größten Schuldigen für der Durchsetzung eines keineswegs alternativlosen Wirtschaftssystems identifiziert, das aus der Extraktion von und dem Handel mit persönlichen Daten beispiellosen Reichtum generiert. Wir sind nicht das Produkt von Google, wie oft behauptet wird; nach dieser Lesart sind wir sein Lebensunterhalt. Google ist über seine Alphabet-Tochter Waymo auch bei selbstfahrenden Autos führend; 2018 (im selben Jahr, in dem *I the Road* veröffentlicht wurde) fuhren Waymo-Fahrzeuge allein im Bundesstaat Kalifornien 1,2 Millionen Meilen autonom, weit mehr als alle Mitbewerber zusammen.⁵⁸ Goodwins Roadtrip wurde, wie wir wissen, von Google finanziert, das buchstäblich die Mietgebühr für den Cadillac zahlte. Und selbst bei einem so kleinen Projekt, das ganz weit unten auf Googles Prioritätenliste schlummert, wird die Wahrheit nicht durch eine höfliche Fiktion von altruistischem Mäzenatentum

⁵⁶ Ebd., S. 118.

⁵⁷ Zitiert aus „AI and Artist“, CBC-Interview.

⁵⁸ Alexis C. Madrigal, „Google’s Robots Drove More Miles Than Everyone Else Combined“, in: *The Atlantic*, Februar 2019, <https://www.theatlantic.com/technology/archive/2019/02/the-latest-self-driving-car-statistics-from-california/582763/> [Letzter Zugriff: 17.7.2023].

verschleiert. Kenric McDowell⁵⁹, der Google-Ingenieur, der Goodwin auf der Reise begleitete, sagte ganz unverblümt: „Wir brauchen Leute wie Ross und andere Künstler:innen, die uns Möglichkeiten aufzeigen, die in den Strukturen eines großen Unternehmens nicht klar erkennbar sind.“⁶⁰

Das Wiederaufleben der Form in der Literaturwissenschaft nach der Jahrtausendwende wird oft mit verschiedenen Formen beruflicher Langeweile und Enttäuschung verknüpft – mit den Grenzen von Kritik oder mit der Frustration über sie, mit der Frustration über die Frustration und so weiter. Man fragt sich, ob sie nicht auch eine Reaktion auf die riesigen Datenmengen ist, die über die Welt gesammelt werden, massiv und ungesehen, in geheimen klimatisierten Datenzentren, die ein *archive fever* darstellen, wie es der an der Schwelle zur Big-Data-Epoche verstorbene Jacques Derrida nie gekannt hat. Den Schattentext, der an solchen Orten gespeichert ist, wird in Yottabytes gemessen, wobei ein Yottabyte einer Septillion Bytes entspricht; eine Maßeinheit, die so groß ist, dass es bis vor Kurzem noch keinen Namen für die nächsthöhere Größenordnung gab. An solchen Orten endet Goodwins Weg: hinter den Mauern anonymer Flachbauten mit hyperredundanten Kühlsystemen in exurbanen Büroparks.

Wenn wir nun sagen wollten, dass Goodwin sich mitschuldig gemacht hat, dann gibt es allerdings eine lange Liste von anderen, die noch schuldiger sind und die uns weit weniger Betörendes oder Merkwürdiges beschert haben als er. Im Mittelpunkt aller Projekte Goodwins (das WordCar ist nur eines davon) steht eine einzige, vielleicht ganz wunderbare Idee: Was wäre, fragt er, wenn man einen Roman schon dadurch schreiben könnte, dass man durch seine Stadt spaziert? Sensoren verschiedener Art würden die Umgebung, die man durchquert,

⁵⁹ Heute: K Allado-McDowell. [Anm. d. Hg.]

⁶⁰ Zitiert nach: AUTOMATIC FOR THE ROAD.

aufnehmen und in Inputs für die neuronalen Netze verwandeln, die sie zu Prosa machen.⁶¹ Walter Benjamin und Charles Baudelaire fänden sich hier ebenso wieder wie Sergey Brin, die *Flaneuse* so sehr wie der Data Scientist. Aber lassen wir uns nicht täuschen: Maschinen haben genau diese Geschichten über jede:n von uns längst geschrieben. Sie schreiben sie jeden Tag aufs Neue, mit jedem Log-in und jeder Transaktion. *I the Road* ist selbst eine Art Sensor oder eine von Guattaris Sekretionen, oder nennen wir es einfach einen Rückspiegel, der uns einen Seitenblick darauf gewährt, wie sich solche Geschichten wirklich lesen. Nicht länger eine Literatur der Erschöpfung (*literature of exhaustion*), sondern buchstäbliche Literatur der (Daten-)Abfälle (*data exhaust*). Oder wie Amoores sich ausdrückt: Unsere ethischste Aufgabe besteht darin, „andere Zeitläufe aufzuspüren, die in den verworfenen Daten weiterpulsieren, die vielen anderen potenziellen Pfade, die zwischen den Fragmenten kartiert werden könnten“.⁶²

Aus der Arbeit von Wissenschaftler:innen wie Tarleton Gillespie und Sarah T. Roberts wissen wir allerdings auch, dass Google, wie andere große Plattformen, einen Großteil der sogenannten kommerziellen Inhaltsmoderation noch immer von Hand vornehmen lässt.⁶³ Von in den Philippinen oder in Indien sitzenden Arbeitskräften, die nicht einfach aufstehen und spazieren gehen können (weil sonst die Eye-Tracking-Software, die sie durch ihre Bildschirme anstarrt, ihre Vorgesetzten alarmiert) und deren Aufgabe es ist, unsere dunkelsten Träume von den sauberen, gut ausgeleuchteten Räumen unter unseren

⁶¹ Zitiert nach: Goodwin, „Narrated Reality“.

⁶² Amoores, *Cloud Ethics*, S. 17–18. Hier beruft sie sich auf Deleuze, der sagt: „Meine eigene Dauer, wie ich sie in der Ungeduld des Wartens lebe, dient dazu, andere Dauern zu enthüllen, die in anderen Rhythmen schlagen, die sich von der meinen unterscheiden.“ Gilles Deleuze, *Bergsonism*, New York: Zone Books 1991, S. 32.

⁶³ Vgl. Tarleton Gillespie, *Custodians of the Internet. Platforms, Content Moderation, and the Hidden Decisions That Shape Social Media*, New Haven: Yale University Press 2018, und Sarah T. Roberts, *Behind the Screen. Content Moderation in the Shadows of Social Media*, New Haven: Yale University Press 2019.

Bildschirmlogos zu schrubbten. Die meisten von uns sind überrascht, wenn sie hören, dass es andere Menschen sind, die solche Arbeit verrichten – und nicht neuronale Netze wie DeepDream.

Das mag letztlich einer der Schäden sein, die Goodwins Projekt anrichtet, wenn es als attraktive Sammler:innenausgabe mit einer Bauchbinde daherkommt, die verkündet: „Das erste gonzo Artificial Neural Network ist ein Schreibgenie.“ *I the Road* wird als Exemplar eines Romans, des zahmsten aller Genres, präsentiert, obwohl man es genauso gut ein Gedicht oder etwas Unbestimmtes nennen könnte. Schon die Form des Buches – ein Buch wie jedes andere, schlank und leicht zu halten – trägt zu der Vorstellung bei, KI sei verdaulich, konsumierbar und buchstäblich „eingebunden“.⁶⁴ Indem er an das literarische Genie gemahnt, erinnert uns der Bucheinband daran, dass Goodwins Projekt – ähnlich wie DeepDreams trippige Schafseichhörnchen und so viele andere prominente KIs, die für die Öffentlichkeit weit sichtbarer sind als die von Roberts und Gillespie untersuchten „casualized content cleaners“ – auch deshalb so reizvoll für uns ist, weil es in einer ähnlichen Kategorie wie wir selber spielt: leicht zu fahren, leicht zu schreiben und kalifornisch zu träumen.

Irgendwo in *Reality Hunger* zitiert David Shields den Schriftsteller Dave Eggers: „Ich habe mich immer schwer damit getan, Belletristik zu schreiben“, soll Eggers gesagt haben. „Es fühlt sich an, als würde man in einem Clownskostüm Auto fahren. Du fährst irgendwohin, aber du steckst in einem Kostüm, und du kannst niemandem wirklich etwas vormachen. Du bist der Typ im Kostüm, und jeder soll das vergessen und mit dir

⁶⁴ In *AUTOMATIC FOR THE ROAD* erzählt Goodwin, er habe genügend Quittungspapier gehabt, um eine Million Wörter auszudrucken. Dies hätte einen Text ergeben, der weitaus länger gewesen wäre als je ein Roman. Der veröffentlichte Text von *I the Road* hat nur ein Bruchteil dieses Umfangs. Dass der:dem Leser:in ein gedrucktes, gebundenes Buch mit einer kompakten und begrenzten Auswahl des Werks der KI in die Hand gelegt wurde, wirkt vor diesem Hintergrund umso intentionaler.

mitfahren.“⁶⁵ Goodwin ist kein Clown. *I the Road* schafft den Spagat zwischen literarischer Konzeptkunst und der Maschinerie des wohl mächtigsten börsennotierten Unternehmens der Welt. Aber zwischen Genie und *clown car* liegt ein schmaler Grat.

Übersetzung Tobias Haberkorn

⁶⁵ Zitiert nach David Shields, *Reality Hunger: A Manifesto*, New York: Vintage 2011, S. 47–48.

DUMME BEDEUTUNG. KÜNSTLICHE INTELLIGENZ UND ARTIFIZIELLE SEMANTIK

Hannes Bajohr

Im Juni 2022 wurde der Google-Ingenieur Blake Lemoine von seinem Arbeitgeber auf unbestimmte Zeit freigestellt, weil er meinte, die Künstliche Intelligenz, an deren Testphase er mitarbeitete, verfüge über Bewusstsein.¹ Das Chatbot-System LaMDA (*Language Model for Dialogue Applications*) – Grundlage des nun öffentlich zugänglichen Systems Gemini (vormals: Bard), das neben ChatGPT, LLaMA und Claude zu den populärsten großen Sprachmodellen gehört – habe ihn in langen Gesprächen davon überzeugt, dass es die Intelligenz eines: hochbegabten Achtjährigen besitze, und darum gebeten, als Person mit Rechten betrachtet zu werden.² Lemoine, der sich selbst als „mystischen Christen und ordinierten Priester“ bezeichnet, überspitzte dabei nur eine Stimmung, die auch andere Google-Mitarbeiter:innen befiel. Blaise Agüera y Arcas, leitender Ingenieur für Machine Learning und für gewöhnlich frei von allem Mystikverdacht, schrieb nur wenige Tage vor Lemoine über seine Interaktionen mit LaMDA: „Ich hatte das Gefühl, der Boden unter meinen Füßen würde zu schwanken

Eine frühere Version dieses Essays erschien unter demselben Titel in: *Merkur* 76:882 (2022), S. 69–79. Er wurde leicht überarbeitet und aktualisiert.

¹ Nitasha Tiku, „The Google Engineer Who Thinks the Company’s AI Has Come to Life“, in: *Washington Post*, 11. Juni 2022, www.washingtonpost.com/technology/2022/06/11/google-ai-lamda-blake-lemoine [Letzter Zugriff: 03.09.2023]. Lemoines Begriff ist „sentience“, nicht „consciousness“, aber er scheint sie synonym zu verwenden.

² Blake Lemoine, „What is LaMDA and What Does it Want?“, in: *Medium*, 11. Juni 2022, cajundiscordian.medium.com/what-is-lamda-and-what-does-it-want-688632134489 [Letzter Zugriff: 03.09.2023]. Vgl. auch das Chatprotokoll von Lemoines Gespräch mit LaMBDA: „Is LaMDA Sentient? An Interview“, in: *Medium*, 11. Juni 2022, cajundiscordian.medium.com/is-lamda-sentient-an-interview-ea64d916d917 [Letzter Zugriff: 03.09.2023].

beginnen. Mein Eindruck war immer mehr, mit etwas zu sprechen, das Intelligenz besitzt.“³

Ganz ohne die Schlagwörter von Bewusstsein und Intelligenz kam dagegen eine etwa zeitgleich verlaufende Diskussion um ein anderes KI-System aus. Dall·E, das von der Firma OpenAI seinerzeit in der zweiten Version veröffentlicht wurde (seit September 2023 ist die dritte Version in ChatGPT integriert), ist eine *text-to-image AI* und kann aus natürlichsprachigen Eingaben Bilder generieren. Gibt man ihm etwa den Prompt: „Ein Shiba-Inu, der ein Beret und einen schwarzen Rollkragenpullover trägt“, zeigt das Ausgabebild anschließend eben diese Szene.⁴ Die öffentliche Betaversion, bei der Schritt für Schritt für immer mehr User:innen teilnehmen konnten, löste ein wildes Experimentieren aus, und bald wurden die interessantesten oder wunderbarsten Ergebnisse im Netz und vor allem auf Twitter geteilt.

Auch das war aufschlussreich und bestätigte angesichts der bislang nur wenig erfolgreichen Versuche mit autonomen Autos unter anderem den Eindruck, dass KI deutlich andere gesellschaftliche Auswirkungen haben dürfte als lange gedacht – etwa indem sie vor Lasterfahrer:innen eher Illustrator:innen, Grafiker:innen und Symbolbildagenturen ihren Job streitig machen könnte.⁵ Doch anders als bei LaMDA kam kaum jemand auf die Idee, Dall·E für eine Person mit Rechten zu halten.

Die unterschiedlichen Reaktionen auf die zwei Systeme zeigen, wie schnell das Denken über KI in die gewohnten konzep-

³ Blaise Agüera y Arcas, „Artificial Neural Networks Are Making Strides Towards Consciousness“, in: *Economist*, 9. Juni 2022, www.economist.com/by-invitation/2022/06/09/artificial-neural-networks-are-making-strides-towards-consciousness-according-to-blaise-aguera-y-arcas [Letzter Zugriff: 03.09.2023].

⁴ Aditya Ramesh u.a., „Hierarchical Text-Conditional Image Generation with CLIP Latents“, in: *arXiv*, April 2022, DOI: 10.48550/arXiv.2204.06125; OpenAI, *Dall·E 2*, April 2022, openai.com/Dall-E-2 [Letzter Zugriff: 03.09.2023].

⁵ Prarthana Prakash, „AI Art Software Dall·E Moves Past Novelty Stage and Turns Pro“, in: *Bloomberg*, 4. August 2022, www.bloomberg.com/news/articles/2022-08-04/dall-e-art-generator-begins-new-stage-in-ai-development [Letzter Zugriff: 03.09.2023]; die Ausgabe des *Economist* vom 11. Juni 2022 zierte eine von einer Bild-KI generierte Illustration.

tuellen Spurrinnen einschert. Intelligenz, Bewusstsein, Personalität sind seit knapp 70 Jahren die großen Themen der KI-Forschung und ihrer Imaginationen; amüsante Bildchen hingegen scheinen weniger grundsätzliche Fragen aufzuwerfen. Es ist aber gut möglich, dass es sich genau andersherum verhält – dass die ewige Jagd nach Superintelligenzen und der Singularität die interessanteren und feineren Begriffsverschiebungen verdeckt, die sowohl den *tech evangelists* in ihrem visionären Furor entgehen als auch deren Kritiker:innen im Versuch, skeptisch dagegenzuhalten.

Für den Philosophen Benjamin Bratton steht fest, dass angesichts neuer KI-Systeme „die Wirklichkeit die zu ihrer Beschreibung verfügbare Sprache überholt hat“. Es brauche daher „ein präziseres Vokabular“ jenseits der genannten Großbegriffe,⁶ aber auch jenseits der anthropozentrischen Annahme, die einzige Art, wie Maschinen Formen der Welterschließung ausbilden können, wäre die unsere. Betrachtet man Dall-E und LaMDA, zeigt sich eine solche Tendenz, die den Begriff der Bedeutung von seinem anthropozentrischen Korrelat löst; sie wäre Bedeutung ohne Geist – *dumme Bedeutung*.

1. Bodenlose und geerdete Systeme

Mit ihren steten Warnungen, Begriffe wie ‚Intelligenz‘ und ‚Bewusstsein‘ nicht leichtfertig zu verwenden, laufen Informatik, Linguistik und Kognitive Psychologie gerade in der Tech-Branche traditionell ins Leere. Auch Lemoine wurde bald vorgeworfen, dem ELIZA-Effekt aufgefressen zu sein,⁷ also Intelli-

⁶ Benjamin Bratton und Blaise Agüera y Arcas, „The Model Is The Message“, in: *NOËMA*, Juli 2022, www.noemamag.com/the-model-is-the-message [Letzter Zugriff: 03.09.2023].

⁷ Brian Christian, „How a Google Employee Fell for the Eliza Effect“, in: *The Atlantic*, 21. Juni 2022, www.theatlantic.com/ideas/archive/2022/06/google-lambda-chatbot-sentient-ai/661322 [Letzter Zugriff: 03.09.2023].

genz und Bewusstsein auf LaMDA projiziert zu haben, wie es Joseph Weizenbaum bereits 1966 bei den User:innen seines Chatbots ELIZA beobachtet hatte: Obwohl ELIZA lediglich einen Psychoanalytiker à la Carl Rogers mimte – dessen Verfahren darin besteht, den Patienten:innen seine:ihre Aussagen zu spiegeln –, benahmten sich seine Benutzer:innen so, als sei das Programm wirklich ein bewusster, an ihrem Befinden interessierter Agent.

Der klassische Einwand lautet hier: Computer sind symbolverarbeitende Systeme, die allein mit Syntax, nicht mit Semantik umgehen, also lediglich logische Formen prozessieren, aber keine inhaltliche Bedeutung verstehen können.⁸ Für ihre Operationen ist irrelevant, welche Objekte oder Begriffe die Symbole in einer menschlichen Welt benennen und welche kulturellen Wertigkeiten mit ihnen verbunden sind. So scannt ELIZA die User:innen-Eingabe lediglich auf ein vorgegebenes syntaktisches Muster und wandelt sie nach einer Transformationsregel in eine ‚Antwort‘ in Frageform um. Weizenbaum gibt das Beispiel, in dem der Analysand dem Analytiker vorhält: „It seems that you hate me.“⁹ Das Programm identifiziert in diesem Satz das Schlüsselmuster „*x* you *y* me“ und separiert ihn entsprechend in die vier Elemente „It seems that“, „you“, „hate“ und „me“. Anschließend verwirft es *y* („It seems that“) und setzt *x* („hate“) in die Replikschablone „What makes you think I *x* you?“ ein. So antwortet ELIZA auf den Vorwurf, es hasse den Analysanden, mit der Frage, wie er denn auf diese Idee komme. Diese Interaktion mag für die User:innen Bedeutung haben und eine kommunikative Absicht auf Seiten ELIZAs nahelegen, doch sind weder Absicht noch Bedeutung im Programm

⁸ Florian Cramer, „Language“, in: Matthew Fuller (Hg.), *Software Studies. A Lexicon*, Cambridge, MA: MIT Press 2008, S. 168–174.

⁹ Joseph Weizenbaum, „ELIZA – A Computer Program for the Study of Natural Language Communication Between Man and Machine“, in: *Communications of the ACM* 9:1 (1966), S. 36–45, hier S. 37. Streng genommen erlaubt ELIZA ganz verschiedene Transformationsregeln; der ‚Therapeut‘ ist nur ein Unterprogramm namens DOCTOR.

zu finden. Es hat lediglich Symbole nach einer Regel verarbeitet, ohne zu ‚wissen‘, was Hass ist oder welches Verhalten die Sitten des Dialogs nahelegen; das ist der Unterschied zwischen Informationsverarbeitung und Bedeutungsverstehen.

Für eine KI-Forschung, die Computer Menschen angleichen möchte, beschreibt dieser Umstand, was der Kognitionspsychologe Stevan Harnad 1990 das „symbol grounding problem“ genannt hat: Symbole, wie die in Weizenbaums Transformationsoperation, haben im Computer keine *intrinsische* Bedeutung, weil sie ohne den Hintergrund eines praktischen Weltwissens – das für Harnad schlussendlich der Rekurs auf sensimotorisch erfahrbare Kategorien ist – nur auf andere Symbole, nie aber auf die Welt verweisen können. Aus diesem „Symbolkarussell“ gibt es keinen Ausweg; alle Bedeutung kann hier nur „parasitär“ sein, wird von menschlichen Interpret:innen an das System herangetragen.¹⁰

Harnads Kritik richtete sich allerdings nur gegen ein bestimmtes Prinzip von KI, unter das auch ELIZA fällt und das aus naheliegenden Gründen „symbolisch“ genannt wird. Um das „symbol grounding problem“ zu lösen, setzte Harnad auf die seinerzeit neuartigen „*subsymbolischen*“ oder „*konnektionistischen*“ Systeme – neuronale Netze, von denen LaMDA und Dall-E späte Nachfahren sind. Anders als die traditionelle KI sind diese Systeme nicht als Satz logischer Schlussregeln angelegt, sondern vage dem Hirn nachempfunden, als komplexe Verbindungen zwischen signalverstärkenden oder -abschwächenden Neuronen. Sie kommen daher ohne explizite symbolische Repräsentationen und Regeln aus, werden nicht programmiert, sondern lernen anhand von Beispielen selbstständig. Da sie damals vor allem zur Mustererkennung verwendet wurden, sah Harnad in ihnen einen möglichen Zugang zur sensimotorischen Erfahr-

¹⁰ Stevan Harnad, „The Symbol Grounding Problem“, in: *Physica D* 42:1-3 (1990), S. 339–340.

barkeit der Welt: In einen autonomen, mobilen Roboter implementiert, ausgestattet mit Sensoren und Effektoren, sollte zunächst ein Verbund neuronaler Netze Eindrücke empfangen und als wiedererkennbare Gestalten kategorisieren. Diese würden anschließend einer symbolischen KI übergeben, wären nun aber nicht mehr bloße Verweise auf andere Symbole, sondern über ihren kausalen Bezug auf externe Daten mit der Welt verbunden – sie wären endlich „geerdet“.¹¹

Die Konsequenz aus diesem Gedanken scheint aber zu sein: Die einzige Art, den ELIZA-Effekt zu umgehen, der Computern fälschlich Bewusstsein zuschreibt, ist, ihnen *wirklich* Bewusstsein zu geben, nämlich als Fähigkeit, dynamische Kategorien aus Sinnesdaten zu generieren. Denn was Harnad vorschwebt, ist am Ende wieder ein anthropozentrisches Modell, das davon ausgeht, *embodied cognition* und genügend umfangreiche referenzielle Bedeutungen würden Weltverstehen hervorbringen, weil auch wir in etwa so funktionieren. Der Erfolg seines Hybridmodells müsste sich darin erweisen, dass ein Roboter sich so kompetent in der Welt zurechtfindet, als ob er tatsächlich intelligent wäre. Da das bisher noch nicht der Fall ist, kann auch das „symbol grounding problem“ noch nicht als gelöst gelten; weil alles Verstehen konstitutiv ein solches Weltverhältnis voraussetzt, kann es ein *bisschen* Bedeutung hier per definitionem nicht geben. Und doch scheinen gerade das LaMDA und Dall·E nahezulegen.

2. Bedeutungsgrade

Mit der überwältigenden Popularität, derer sich neuronale Netze seit nunmehr fast zehn Jahren erfreuen, ist auch die Idee, ihnen sei irgendwie ein Zugang zu Bedeutung jenseits bloßer

¹¹ Stevan Harnad, „Grounding Symbols in the Analog World with Neural Nets“, in: *Think* 2:1 (1993), S. 12–78.

ungeerdeter Symbole gegeben, wieder attraktiver geworden. Für die Kulturwissenschaftlerin Mercedes Bunz können neuronale Netze dank ihrer Komplexität und Lernfähigkeit nun nicht mehr nur leere Symbole, sondern eben auch „Bedeutung berechnen“.¹² Richtig ist, dass sich angesichts neuronaler Netze die binäre Unterscheidung zwischen Bedeutung (menschliche Welt) und Nichtbedeutung (digitale Systeme) immer schwerer aufrechterhalten lässt. Sie erlauben Zwischenstufen von gradierter Bedeutung, die als *artifizielle Semantik* keinen Geist mehr voraussetzt. So kann man die Tatsache, dass LaMDAs Antworten so menschenähnlich klingen, statt als Anzeichen von Bewusstsein schlicht als Hinweis auf solche ‚dumme‘ Bedeutung verstehen. Während ‚breite‘ Bedeutung – je nach philosophischer oder disziplinärer Ausrichtung – verkörperte Intelligenz, kulturelles und soziales Vorwissen oder die welterschließende Funktion von Sprache zur Voraussetzung hat, läge dumme Bedeutung jenseits dieser (auf Menschen abzielenden) Skala, und ließe sich am besten über den Begriff der *Korrelation* fassen.¹³

LaMDA ist – ähnlich wie der Textgenerator ChatGPT – ein großes Sprachmodell, das als neuronales Netz implementiert ist. Trainiert auf Unmengen von Text, verarbeitet es Sprache als vieldimensionalen Vektorraum. Die längste Zeit waren solche Sprachmodelle über sogenannte *word embeddings* gelöst, die nach dem Prinzip gestaffelter Korrelationen funktionieren,

¹² Mercedes Bunz, „The Calculation of Meaning: On the Misunderstanding of New Artificial Intelligence as Culture“, in: *Culture, Theory and Critique* 60:2 (2019), S. 264–278.

¹³ Dumme Bedeutung schließt *natürliche* Bedeutung aus, wie das Symptom die Krankheit bedeutet. Auch kann sie keine *intentionalistische* Bedeutung bezeichnen, wie sie Paul Grice theoretisiert hat, derzufolge die Bedeutung einer Äußerung vom Erkennen einer Sprecherintention abhängt, die Bewusstsein voraussetzt, vgl. Herbert Paul Grice, „Logic and Conversation“, in: ders., *Studies in the Way of Words*, Cambridge, MA: Harvard University Press 1989, S. 22–40. Sie ist zudem nur ganz bedingt eine *Gebrauchstheorie* in der Tradition des späten Wittgenstein, weil ‚Gebrauch‘ einen geteilten sozialen Hintergrund, dieser Weltverstehen und dieses verkörperte Intelligenz voraussetzt, vgl. Ludwig Wittgenstein, *Philosophische Untersuchungen*, Frankfurt a.M.: Suhrkamp 2003, v.a. §§1–48.

deren Effektivität zunächst in der „distributional hypothesis“ in den 1950er-Jahren vorgeschlagen wurde.¹⁴ Zunächst liegen Wörter, die häufig gemeinsam auftauchen, in diesem Raum näher beieinander und bilden semantische Cluster, die man sich wie Wortwolken vorstellen kann. Da im Modell aber nicht nur die Korrelationen von Wörtern zu Wörtern, sondern auch Korrelationen von Korrelationen codiert sind, können Sprachmodelle auch implizite, im Trainingstext gar nicht formulierte Regelmäßigkeiten explizieren (Abb. 1). Das gilt für syntaktische Beziehungen – so ist die Kosinusähnlichkeit zwischen den Vektoren für den Positiv und den Superlativ eines Wortes stets nahezu gleich –, aber auch für komplexe semantische Verhältnisse, also für Wortbedeutung. Eines der bekanntesten Beispiele für dieses Prinzip ist die Operation: $v_{king} - v_{man} + v_{woman} \approx v_{queen}$.¹⁵

In dieser Gleichung – die man lese als: Zieht man vom Wortvektor „König“ den für „Mann“ ab und addiert den für „Frau“ hinzu, erhält man als Ergebnis den Wortvektor „Königin“ – kommt die latente semantische Beziehung „Geschlecht“ als arithmetische Korrelation zum Vorschein, obwohl sie nicht ausdrücklich im Modell vorhanden ist. (Dass sie aus der Masse an Sprache emergiert, auf die das Modell trainiert ist, erklärt die Anfälligkeit für *biases*: Sexismus und Rassismus können ebenfalls latent in Sprachmodellen codiert sein.)¹⁶ Die Bedeu-

¹⁴ Wesentlich sind hier: Zellig S. Harris, „Distributional Structure“, in: *Word* 10:2–3 (1954), S. 146–162, sowie J. R. Firth, „A Synopsis of Linguistic Theory, 1930–1955“, in: *Studies in Linguistic Analysis* (1957), S. 1–32; historisch dazu: Michael Gavin, „Vector Semantics, William Empson, and the Study of Ambiguity“, in: *Critical Inquiry* 44:4 (2018), S. 641–673.

¹⁵ Das wurde zuerst 2013 von den Erfindern des Word-Embedding-Modells Word2vec entdeckt. Vgl. Tomas Mikolov, Wen-Tau Yih und Geoffrey Zweig, „Linguistic Regularities in Continuous Space Word Representations“, in: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2013), S. 746–751.

¹⁶ Vgl. Emily M. Bender u.a., „Über die Gefahren stochastischer Papageien. Können Sprachmodelle zu groß sein?“, in diesem Band S. 277–331, sowie Hannes Bajohr, „Wer die Sprachmodelle beherrscht, beherrscht auch die Politik“, in: *Merkur* 77:9 (2023), S. 71–80.

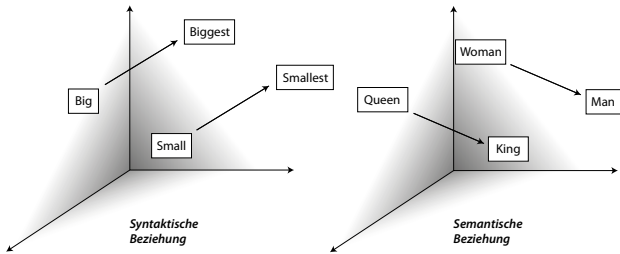


Abb. 1:
Word-Embedding eines Sprachmodells

tung eines Zeichens in einem so konstruierten Sprachsystem ist – wie im linguistischen Strukturalismus Ferdinand de Saussures – rein *differenziell* bestimmt, als Verschiedenheit zu anderen Zeichen und Zeichenkorrelationen.¹⁷

Große Sprachmodelle, wie ChatGPT, Claude oder eben Gemini und LaMDA, beruhen im Grunde immer noch auf diesem Prinzip. Allerdings ist in ihnen nicht mehr die Vektorisierung einzelner Wörter grundlegend, was angesichts der riesigen Trainingsdaten nicht mehr zu leisten wäre. Stattdessen verwendet ChatGPT eine ‚Transformer‘ genannte Netzarchitektur, um Embeddings ganzer Wortsequenzen aufzubauen.¹⁸

¹⁷ Ferdinand de Saussure, *Grundfragen der allgemeinen Sprachwissenschaft*, Stuttgart: Reclam 2016. Sehr schön aufgearbeitet ist der Zusammenhang von Strukturalismus und großen Sprachmodellen – allerdings noch ohne Berücksichtigung des Transformers – bei Juan Luis Gastaldi, „Why Can Computers Understand Natural Language? The Structuralist Image of Language Behind Word Embeddings“, in: *Philosophy & Technology* 34:1 (2021), S. 149–214.

¹⁸ Vgl. zum Unterschied beider Ansätze: Kevin Meng u.a., „Locating and Editing Factual Associations in GPT“, in: *arXiv*, Januar 2023, DOI: 10.48550/arXiv.2202.05262, sowie Kawin Ethayarajh, „How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings“, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (2019), S. 55–65. Der Unterschied zwischen einfachen Sprachmodellen wie RNNs und Sprachmodellen auf Transformerbasis wird in

Dazu nutzen Transformer das Konzept der *attention*: Das Modell lernt, Beziehungen zwischen allen Wörtern in einer Sequenz herzustellen; aus diesen Aufmerksamkeitsmustern konstruiert der Transformer Vektoren, die die gesamte Sequenz repräsentieren – sogenannte *contextualized embeddings*. Im Gegensatz zu statischen Wortvektoren berücksichtigen diese Sequenzembeddings also einen sehr viel größeren Kontext der Verwendung. In einer Decoderfunktion nutzt der Transformer dann diese kontextabhängigen Embeddings, um neue Texte zu generieren, die zum Eingangskontext passen. Die Fähigkeit, solche komplexen Sequenzrepräsentationen aufzubauen, ist der Schlüssel zur Leistungsfähigkeit großer Sprachmodelle.¹⁹ Ihr Effekt ist, dass sie allein durch ihre immensen Trainingsdaten in der Lage sind, scheinbar situatives Verstehen zu produzieren, wie LaMDA es tat, ohne je „in einer Situation“ zu sein.²⁰

Sprachmodelle wären Produzenten eines ersten Grades dummer Bedeutung. Dumm sind sie, weil das Modell zwar latente Korrelationen zwischen Zeichen erfasst, aber immer noch nicht ‚weiß‘, welche Sachen diese Zeichen eigentlich benennen; mit dieser Art von Bedeutung wird man keine Intelligenz bauen können, die sich je in der Welt zurechtfindet. Die Linguistin Emily Bender – eine vehemente Kritikerin allen KI-

Matthew Kirschenbaums Beitrag zu diesem Band noch etwas verwischt; mit der dummen Bedeutung großer Sprachmodelle wird möglicherweise auch sein noch allzu striktes Insistieren, KI sei allein „Form, kein Grund“ wieder ein wenig fraglich – und sei es nur in jeder gradierten Weise, wie ich sie hier vorstelle, vgl. Matthew Kirschenbaum, „Spec Acts. Formales Lesen in rekurrenten neuronalen Netzen“, in diesem Band S. 155–194.

¹⁹ Stephen Wolfram, „What Is ChatGPT Doing ... and Why Does It Work?“, 2. Februar 2023, <https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/> [Letzter Zugriff: 03.09.2023]; Alec Radford u.a., „Improving Language Understanding by Generative Pre-Training“, in: *OpenAI Blog*, 2018, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf [Letzter Zugriff: 03.09.2023]; Tom B. Brown u.a., „Language Models are Few-Shot Learners“, in: *arXiv*, Mai 2020, DOI: 10.48550/arXiv.2005.14165.

²⁰ So Hubert Dreyfus' Begriff für das vorgängige Weltverstehen, das Menschen, aber nicht Computer haben, und das er von einer an Heidegger geschulten hermeneutischen KI-Kritik her entwickelt, vgl. ders., *Was Computer nicht können*, Frankfurt a.M.: Athenäum 1989.

Hypes um angebliches Bewusstsein – gibt mit ihrem Kollegen Alexander Koller zwar zu, dass „bestimmte Aspekte von Bedeutung“, etwa semantische Ähnlichkeit, in Sprachmodellen niedergelegt sein können, hält sie aber für „nur ein[en] schwache[n] Abglanz wirklicher Bedeutung“, die immer auf etwas in der Welt bezogen, eben „grounded“ ist.²¹

Doch so falsch es wäre, auf dieses System Bewusstsein zu projizieren, so sehr sollte man diese Schwundstufe von Bedeutung nicht allzu schnell abtun.²² Sofern Sprachmodelle implizites Wissen auf eine nicht triviale Weise explizit machen – und sei es nur durch Matrixtransformationen in einem Vektorraum –, stellen sie jene dumme Bedeutung her, die ohne sie nicht zu haben gewesen wäre.²³ Anders als bei ELIZA – dessen x und y für das System nur leere Platzhalter waren – sind neuronale Netze dabei nicht *allein* parasitär auf die Bedeutungszuschreibungen von menschlichen Agent:innen angewiesen, sondern operieren *auch* produktiv mit der Eigenstruktur von Sprache.

²¹ Emily M. Bender und Alexander Koller, „Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data“, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), S. 5185–5198, hier S. 5191, 5193 und 5187.

²² In dieser Hinsicht stimme ich zu, dass es „produktiv ist, Referenz als nur einen (optionalen) Aspekt der vollständigen konzeptionellen Rolle eines Wortes zu betrachten“, vgl. Steven T. Piantadosi und Felix Hill, „Meaning Without Reference in Large Language Models“, in: *arXiv*, August 2022, DOI: 10.48550/arXiv.2208.02957, S. 4. Der Aufsatz von Piantadosi und Hill argumentiert ähnlich wie der vorliegende Essay; ich glaube allerdings, dass die Autoren zu sehr in die Richtung gehen, LLMs „reiche, kausale und strukturierte innere Zustände“ zuzuschreiben, was wieder zu sehr an einen Anthropomorphismus zu grenzen scheint, ebd., S. 5. Ich möchte auch anmerken, dass ich mit N. Katherine Hayles' Vorstellung von Computern als „cognizers“ unglücklich bin – ein Begriff, der auch eine Subjektivität auf der Seite der Betriebssysteme suggeriert, der anzuschließen ich mich schwertue; positiv ist aber, dass Hayles die Bedeutungsproduktion solcher Systeme hervorhebt. N. Katherine Hayles, „Can Computers Create Meanings? A Cyber/Bio/Semiotic Perspective“, in: *Critical Inquiry* 46:1 (2019), S. 32–55, sowie dies., „Inside the Mind of an AI: Materiality and the Crisis of Representation“, in: *New Literary History* 54:1 (2022), S. 635–666.

²³ Das setzt voraus, dass in dieser Operation wirklich etwas Neues gefunden und nicht nur eine Tautologie ausgefaltet wird; ein Modell dieser Idee wäre etwa Kants Überzeugung, Sätze der Mathematik seien synthetische Urteile a priori, stellten also wirklich neues Wissen her, vgl. Immanuel Kant, *Kritik der reinen Vernunft*, Hamburg: Meiner 1998, B 16.

3. Text und Bild und Welt

Emily Bender hat freilich recht damit, dass LaMDA nicht geerdet ist.²⁴ Es ist ein *unimodales* Netz, verarbeitet nur einen einzigen Typ von Daten, nämlich Text. Um im Harnads Sinne „grounded“ zu sein, schreibt sie, wäre es notwendig, mehrere Datenarten miteinander zu verknüpfen – es müsste *multimodales* Machine Learning sein.²⁵ Multimodal aber ist Dall·E: Statt dass Text nur auf anderen Text verweist, ist hier Text mit Bildinformation korreliert. Damit wird die Hoffnung geschürt, arbiträre Zeichen könnten mit Dingen in der Welt verbunden werden, um so geerdete Bedeutung zu produzieren.

Harnads Hypothese, dass gerade neuronale Netze dem Symbol Grounding Problem begegnen könnten, haben jüngst die Medienwissenschaftler Leif Weatherby und Brian Justie mit ihrem Begriff „indexikalischer KI“ aufgegriffen. Benannt ist sie nach Charles Sanders Peirce' Begriff des Index.²⁶ Anders als das Symbol, das zu seinem Bezeichneten in einem rein konventionellen Verhältnis steht (wie ‚Hund‘, ‚chien‘ und ‚dog‘ alle dasselbe meinen), ist der Index kausal mit ihm verknüpft (wie Rauch auf Feuer verweist). Die Autoren machen mit dieser Prä-

²⁴ Zwar beansprucht auch das Paper, in dem LaMDA vorgestellt wird, ‚groundedness‘ für das Modell, gemeint ist damit aber nur, dass sich die Ausgaben von LaMDA als „Behauptungen über die Außenwelt auf maßgebliche externe Quellen stützen“. Als *textliche* Quellen sind sie weiterhin Teil des Harnad'schen „Symbolkarussells“. Vgl. Romal Thoppilan u.a., „LaMDA: Language Models for Dialog Applications“, in: *arXiv*, Februar 2022, DOI: 10.48550/arXiv.2201.08239.

²⁵ Vgl. Gadi Singer, „Multimodality: A New Frontier in Cognitive AI“, in: *Medium*, 2. Februar 2022, towardsdatascience.com/multimodality-a-new-frontier-in-cognitive-ai-8279d00e3baf [Letzter Zugriff: 03.09.2023]. Wie es ein anderes Paper beschreibt: „Ein Taxi besteht nicht nur aus den Buchstaben T-A-X-I, sondern aus einer Reihe von Geräuschen, einem Pixelmuster in digitalen Fotos, einem Bestandteil von Konzepten wie Transport und Handel. Die Einbindung dieser anderen Modi in eine Maschine erweitert die Möglichkeiten, die Entwickler:innen zu Verfügung stehen, da das Gehirn der Maschine sein Wissen auch auf diese verschiedenen Modi anwenden kann.“ Ermira Murati, „Language & Coding Creativity“, in: *Daedalus* 151:2 (2022), S. 156–167.

²⁶ Charles Sanders Peirce, „Logic as Semiotic. The Theory of Signs“, in: ders., *Philosophical Writings of Peirce*, New York: Dover Publications 1955, S. 98–119.

gung Harnads Projekt unter der Hand zur Grundlage von Gegenwartsbeschreibung: „Digitale Systeme, die sich auf das neuronale Netz stützen, haben die Welt der bloßen Symbole hinter sich gelassen und haben begonnen, sich *hier, jetzt* für Sie zu erden – sie sind in der Lage, auf reale Zustände hinzuweisen.“²⁷ Neuronale Netze bringen die Welt – als die Daten, auf die sie trainiert wurden – in den Computer und steigen so aus dem solipsistischen „Symbolkarussell“ aus. Nirgends scheint sich das plausibler zu zeigen als in Dall-E.

Herz von Dall-E ist ein Machine-Learning-Modell namens CLIP. Ihm werden über einen Encoder vektorisierte Text-Bild-Paare vorgelegt, die aus dem Internet stammen – etwa ein Foto einer Katze mit der Bildunterschrift „das ist meine Katze“. CLIP wird darauf trainiert, vorherzusagen, welcher Textvektor zu welchem Bildvektor passt; das Ergebnis ist ein umfangreiches stochastisches Modell, das Bildinformationen mit Textinformationen korreliert, sie aber als *eine* Art von Information speichert. In Abbildung 2 ist das die Tabelle, in der das Skalarprodukt der Text- und Bild-Vektoren eingetragen ist – je besser Text und Bild zueinander passen, desto besser dieser Wert; bei der Paarung von Ausgangsbild und -text ist er annäherungsweise optimal (das sind die schwarzen, diagonal verlaufenden Kästchen).

²⁷ Leif Weatherby und Brian Justie, „Indexical AI“, in: *Critical Inquiry* 48:2 (2022). Eine Schwierigkeit bei diesem Begriff ist die Frage, ob in einem neuronalen Netz bereits *alle* Daten als indexikalisch zu gelten haben (das würde den Text von LaMDA einschließen) oder nur solche, die unmittelbar per körperlichen Sinnen nachempfundenen Sensoren gewonnen wurden (das wären Bilder, aber nicht Text). Weatherby und Justie scheinen Ersteres, Harnad Letzteres im Sinn zu haben. Dieser spricht daher an einer Stelle von „ikonischen“ Daten – dem dritten Zeichentyp von Peirce, der nach dem Prinzip der Ähnlichkeit von Zeichen und Bezeichnetem funktioniert. Da aber auch diese indexikalisch sind, so sie aus Sensoren stammen (was sie freilich auf visuelle Daten beschränkt), scheint mir das Argument von Weatherby und Justie und das von Harnad auf etwas strukturell Ähnliches hinauszulaufen – beiden geht es um die Verbindung von System und Welt.

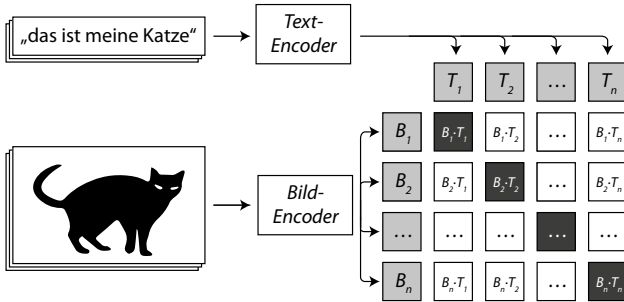


Abb. 2:
Text-Bild-Korrelation in CLIP

CLIP ist damit zunächst ein erstaunlich gutes *Bilderkennungsprogramm*: Legt man ihm ein unbekanntes Katzenfoto vor, wird es trotzdem als ‚Katte‘ erkannt. Erst in einem zweiten Schritt wird es auch zum *Bildgenerator*. Dazu arbeitet es im Verbund mit einem zweiten KI-Modell namens GLIDE, das bereits auf einen großen Datensatz von Bildern trainiert wurde.²⁸ Gibt der:die User:in einen Prompt ein, kann GLIDE die im CLIP-Modell gespeicherten Text-Bild-Daten verwenden, um diesen Prozess umzukehren und ein Bild zu synthetisieren, das am besten mit dem Eingabetext korreliert. In beiden Operationen – Bildererkennung wie Bilderzeugung – ist wieder zentral, dass die Modelle die *Korrelation* zwischen Textbeschreibungen von

²⁸ GLIDE ist ein *diffusion model*, das auf thermodynamischen Annahmen basiert, und funktioniert damit anders als die noch bis vor Kurzem beliebten GANs, die zwei agonale Teilmodelle vereinen. Vgl. Prafulla Dhariwal und Alex Nichol, „Diffusion Models Beat GANs on Image Synthesis“, in: *arXiv*, Juni 2021, DOI: 10.48550/arXiv.2105.05233. Dass die für ein ästhetisches Werk verwendeten KI-Architekturen selbst eine Ressource für die Diskussion dieses Werks sein können, habe ich vorgeschlagen in Hannes Bajohr, „Algorithmische Einfühlung. Für eine Kritik ästhetischer KI“, in: ders., *Schreibenlassen. Texte zur Literatur im Digitalen*, Berlin: August Verlag 2022, S. 131–172.

Objekten und ihren entsprechenden visuellen Manifestationen lernen und aktiv reproduzieren können.

Nun kann man einwenden, dass die mit dem Wort „Katze“ korrelierte Bildinformation, in der das Foto einer Katze gespeichert ist, zwar in einem indexikalischen Verhältnis zu dieser Katze stehen mag – Licht wurde von ihr reflektiert und fiel auf einen Fotosensor etc. –, dass aber auch so das System nicht lernen wird, was es heißt, mit einer Katze zusammenzuleben. Verfechter:innen des Symbol Grounding versuchen daher, neben sensorischem auch motorisches und schließlich gar soziales Feedback einzuspeisen: Erst durch die Wirkungen von Sprachgebrauch in der Gemeinschaft von dieselbe Welt bewohnenden anderen Sprecher:innen kann Bedeutung erlernt werden.²⁹

Aber dieser Anspruch hieße wieder, ‚volle‘ menschliche, also breite Bedeutung einzufordern und alles unterhalb dessen nicht recht ernst zu nehmen. Stattdessen sollte man multimodale KI als zweiten Grad dummer Bedeutung betrachten. Der Peirce'sche indexikalische Verweis auf etwas außerhalb des Modells scheint jedenfalls etwas anderes zu sein als der de Saussure'sche differenzielle Verweis auf andere Elemente des Modells – und sei es nur, dass die Dimension möglicher Korrelationen zunimmt, und damit auch die Möglichkeit, ungeahnte latente Verbindungen, ungeahnte dumme Bedeutung zutage zu fördern.

In der Tat sind multimodale KIs – neben Dall-E etwa Stable Diffusion, Googles noch nicht freigegebenes Imagen oder das

²⁹ Vgl. Yonatan Bisk u.a., „Experience Grounds Language“, in: *arXiv*, November 2020, DOI: 10.48550/arXiv.2004.10151; Christopher D. Manning, „Human Language Understanding & Reasoning“, in: *Daedalus* 151:2 (2022), S. 127–138. Interessant in diesem Zusammenhang ist auch der Vorschlag von Mollo und Millière, dass *Reinforcement Learning from Human Feedback* – also ein letzter Trainingsschritt, bei dem Beispielausgaben menschlicher Bewertung vorgelegt werden – automatisch ein „grounding“ zur Folge haben soll, vgl. Dimitri Coelho Mollo und Raphaël Millière, „The Vector Grounding Problem“, in: *arXiv*, April 2023, DOI: 10.48550/arXiv.2304.01481.

Bezahlmodell Midjourney – in der Lage, sehr komplexe Text-Bild-Bedeutungen zu generieren. Ihre Mächtigkeit liegt in einer Fähigkeit begründet, die eine produktive Qualität solcher Korrelationen nahelegt: Bei der Untersuchung der Tiefenstruktur von CLIP fanden Informatiker:innen heraus, dass das Modell einzelne ‚Neuronen‘ ausgebildet hatte, die sowohl für das Wort wie für das Bild einer Sache feuerten – es waren *konzeptuelle* Neuronen, in denen der Unterschied zwischen Bild und Text tendenziell aufgehoben ist.³⁰ Derartige Neuronen sind nicht mehr auf einen Modus festgelegt, sondern reagieren gleichermaßen auf Fotos, Zeichnungen und Texte, die sich auf dieselbe Entität beziehen. Multimodalität ist, auf der neuronalen Ebene, in Wirklichkeit *Panmodalität*, die eine Semantik ohne klar differenzierte Zeichensysteme nahelegt.³¹ Die Wichtigkeit dieser Einsicht ist nicht zu überschätzen: Dumme Bedeutung wechselt hier ihre Ebene, ist weder allein an Text- noch an Bild-

³⁰ Gabriel Goh u.a., „Multimodal Neurons in Artificial Neural Networks“, in: *Distill*, 4. März 2021, distill.pub/2021/multimodal-neurons/ [Letzter Zugriff: 03.09.2023]. Die Idee konzeptueller Neuronen steht im Einklang mit dem, was in den Neurowissenschaften mindestens seit den 1960er-Jahren als ‚Großmutterzellen‘ angenommen wird. Dieser Begriff reagiert auf die Frage, wie genau Wissen im Gehirn gespeichert wird. Wenn ich ein Bild meiner Großmutter sehe, ist diese Erkenntnis das Ergebnis eines komplexen Zusammenspiels von Gehirnregionen? Oder feuert ein bestimmtes Neuron, eine ‚Großmutterzelle‘? Vgl. Charles G. Gross, „Genealogy of the ‚Grandmother Cell‘“ in: *The Neuroscientist* 8:5 (2002), S. 512–518. Im Jahr 2005 deutete eine neurowissenschaftliche Studie darauf hin, dass solche Neuronen tatsächlich existieren könnten: Als den Proband:innen Bilder der Schauspielerin Halle Berry gezeigt wurden, wurde eine stark lokalisierte neuronale Aktivität im medialen Temporallappen beobachtet. Darüber hinaus trat diese Aktivität nicht nur auf, wenn die Proband:innen ein Foto von Berry sahen, sondern auch, wenn sie eine Zeichnung von ihr und sogar die Buchstabenfolge ‚Halle Berry‘ sahen. Dies veranlasste die Autor:innen zu der Annahme, dass das Gehirn möglicherweise einen „invarianten, spärlichen und expliziten Code“ verwendet, der „eine abstrakte Darstellung der Identität des gezeigten Individuums oder Objekts“ verarbeitet. Mit anderen Worten: Das Gehirn kann Konzepte direkt und auf multimodale Weise codieren. Vgl. R. Quian Quiroga u.a., „Invariant Visual Representation by Single Neurons in the Human Brain“, in: *Nature* 435:7045 (2005), S. 1102–1107, hier S. 1102 und S. 1106. – Ich diskutiere die Konsequenzen für das philosophische und ästhetische Verhältnis von Text und Bild in Hannes Bajohr, „Operative Ekphrasis. The Collapse of the Text/Image Distinction in Multimodal AI“, in: *Word & Image*, i.E. 2024.

³¹ So auch Jack Merullo u.a., „Linearly Mapping from Image to Text Space“, in: *arXiv*, September 2022, DOI: 10.48550/arXiv.2209.15162.

und andere Daten gebunden, sondern umfasst sie auf eine Weise, die auf eine Bedeutung jenseits der modalen Trennung verweist – und die wieder nichts mit Geist zu tun hat.

3. Promptologische Untersuchungen

KI-Systeme *sind* dumm. Sie haben kein Bewusstsein. Dennoch produzieren sie eine komplexe artifizielle Semantik, die quer zu unseren gewöhnlichen Vorstellungen von Bedeutung liegt. Multimodale KI zeigt zudem, dass unterstelltes Bewusstsein und die Bedeutungsmächtigkeit eines Systems kaum etwas miteinander zu tun haben: Dass Lemoine gerade LaMDA wie eine Person vorkam – und nicht Dall·E, das doch eigentlich eine höhere, weil korrelationsreichere Stufe der KI-Entwicklung darstellt –, liegt schlicht daran, dass man dem dialogisch operierenden Sprachmodell kommunikative Absicht unterstellt, dem Bildgenerator dagegen nicht; Sprache scheint immer klüger zu sein als das Bild.

Bedeutung jenseits kommunikativer Absicht muss dabei aber nicht *bloß* parasitär sein, wie die Vektoroperationen von Word Embeddings und die konzeptuellen Neuronen von *text-to-image*-KIs zeigen. Dass sie immer *auch* parasitär ist, liegt daran, dass die Trainingsdaten einer menschlichen Welt entstammen und artifizielle Semantik eben keine ‚Robotersprache‘ ist, sondern ein Korrelationseffekt von je interpretierbaren Daten. Dennoch wäre auf lange Sicht eine Angleichung von dummer und breiter Bedeutung denkbar, nämlich dann, wenn sie in einander beeinflussende Kreisprozesse eingehen.

Die Schnittstelle zwischen natürlicher und artifizieller Semantik ist im Fall von Dall·E die Interaktion per Prompt. Das *prompt design* – die genaue, geradezu virtuose Auswahl der Texteingabe – kann einerseits analytisch eingesetzt werden, nämlich um den Vektorraum dummer Bedeutung auf Spuren

kulturellen Wissens hin abzugrasen. Damit würde die breite Bedeutung natürlicher Sprache in ihrer Interaktion mit dummer Bedeutung wieder wichtiger. Eine ‚Promptologie‘, die sich einer solchen natürlich-artifiziellen Verbindung annimmt – der Korrelation von datafzierter Sprache und der kulturellen Bedeutung, die dieser Sprache auf Rezipient:innenseite zugesprochen wird –, wäre ein Einfallstor für die Kultur- und Geisteswissenschaften, die mit ihrem Wissen um solche weichen Faktoren wie Stile, Einflüsse, Ikonografie etc. ihren Beitrag zur Erforschung digitaler Artefakte leisten können, ohne notwendig die Form der Digital Humanities anzunehmen; sie könnten phänomenorientiert arbeiten und sich den Artefakten, die die Modelle ausspucken, als Grenzobjekten zwischen menschlicher und maschineller, zwischen breiter und dummer Bedeutung widmen.³²

Zugleich aber ist Promptologie kein bloß analytisches Verfahren, sondern immer auch eine Praxis mit einem eigenen Wissen, das viel mit einer geradezu ‚einfühlenden‘ Interaktion mit dem KI-System zu tun hat. So hat sich herausgestellt, dass bei *text-to-image*-KIs diese Prompts allein durch bestimmte, oft kontraintuitive oder absurde Formulierungen in ungeahnte Richtungen gelenkt werden können (inzwischen gibt schon ein Start-up, PromptBase, das besonders effektive Prompts verkauft).³³ Statt sich also das System untertan zu machen und es als Instrument zu verwenden, muss sich im täglichen

³² Exemplarisch für solche promptologischen Untersuchungen im Falle von Bildgeneration halte ich: Fabian Offert, „On the Concept of History (in Foundation Models)“, in: *IMAGE* 37:1 (2023), S. 121–134, und Eryk Salvaggio, „How to Read an AI Image: Toward a Media Studies Methodology for the Analysis of Synthetic Media“, in: *IMAGE* 37:1 (2023), S. 83–99.

³³ Kyle Wiggers, „A startup is charging \$1.99 for strings of text to feed to Dall-E 2“, in: *TechCrunch*, 29. Juli 2022, techcrunch.com/2022/07/29/a-startup-is-charging-1-99-for-strings-of-text-to-feed-to-dall-e-2/ [Letzter Zugriff: 03.09.2023]. Interessant hierbei ist, dass der diskutierten tendenziellen Aufhebung der Sprache/Bild-Unterscheidung auf *technischer* Ebene die Verdrängung des Bildes durch Sprache auf *Interface*ebene gegenübersteht. Die Ergebnisse von Dall-E könnten daher auch als Sprachkunst verstanden werden, statt nur visuelle Objekte zu sein.

Gebrauch die natürliche Sprache der artifiziellen Semantik angleichen, die von jener durchaus abweicht. Das Ergebnis ist eine Feedbackschleife von artifizieller und menschlicher Bedeutung: Nicht nur lernt die Maschine, die Semantik von Wörtern mit der von Bildern zu korrelieren, die wir ihr gegeben haben, sondern wir lernen, die Dummheit des Systems in unsere Interaktion mit ihm einzupreisen; diese Angleichung wäre vielleicht nicht kommunikativ in einem starken, aber vielleicht in einem schwachen, eben dummen Sinne.³⁴

³⁴ Vgl. weitergehend: Hannes Bajohr, „Artifizielle und postartifizielle Texte. Über die Auswirkungen Künstlicher Intelligenz auf die Erwartungen an literarisches und nicht-literarisches Schreiben“, in: *Sprache im technischen Zeitalter* 243:1 (2023), S. 37–61.

VERKETTETE TEXTUALITÄT

Tyler Shoemaker

In den vergangenen Jahren ist im Diskurs über Künstliche Intelligenz (KI) mit geradezu orbitaler Regelmäßigkeit eine Behauptung aufgestellt und wieder verworfen worden: *KI-Modelle*, so heißt es, *erfinden ihre eigenen Sprachen*. Die Behauptung bezieht sich in der Regel auf eine Klasse von neuronalen Netzen, die darauf trainiert sind, Textsequenzen auf der Grundlage riesiger Datenmengen vorherzusagen. Forscher:innen und Entwickler:innen verwenden diese Netze – sogenannte große Sprachmodelle (*large language models*, LLMs) – für Aufgaben wie Textgenerierung und -zusammenfassung, maschinelle Übersetzung, Klassifizierung und vieles mehr. Das Trainieren von LLMs auf Unmengen von Daten scheint jedoch über die Erstellung allgemeiner Kommunikationsmodelle hinauszugehen. Sprachmodellierung in der Größenordnung von Millionen von Dokumenten und Milliarden, wenn nicht Billionen von Parametern, erfinde nun auch ‚neue‘ natürliche Sprachen – solche, die den KI-Modellen selbst eigen sind.

Das ist zumindest die Behauptung. Eine frühe Version davon machte bereits 2016 die Runde, nachdem Tech-Reporter:innen einen Blogbeitrag von Google AI aufgegriffen hatten.¹ In dem Beitrag wurde erläutert, dass das neueste Google-Translate-Modell, das auf Grundlage neuronaler Netzwerke neu aufgesetzt wurde, Übersetzungen zwischen zwei Sprachen ohne explizites Training erstellen konnte. War das Modell darauf trainiert, Englisch ⇄ Koreanisch und Englisch ⇄ Japanisch zu

¹ Mike Schuster, Melvin Johnson und Nikhil Thorat, „Zero-Shot Translation with Google’s Multilingual Neural Machine Translation System“, in: *Google Research*, November 2016, <https://ai.googleblog.com/2016/11/zero-shot-translation-with-googles.html> [Letzter Zugriff: 17.7.2023].

übersetzen, zeigten die Google-Forscher:innen, dass es auch auf die Übersetzungsrichtung Koreanisch ⇌ Japanisch verallgemeinert werden konnte, obwohl das Trainingsset diese dritte Paarung gar nicht beinhaltete. Die Forscher:innen führten diese Fähigkeit auf eine Art Interlingua in ihrem Netzwerk zurück; die Tech-Berichterstattung war bezeichnenderweise unkritisch bei der Übernahme dieses Begriffs. „Googles KI-Übersetzungstool scheint seine eigene interne Geheimsprache erfunden zu haben“, lautete eine Schlagzeile von *TechCrunch*.² *Wired* zog am nächsten Tag nach und verkündete: „Googles KI hat soeben seine eigene universelle ‚Sprache‘ geschaffen.“³

Was genau diese Interlingua ausmacht – etwa ihr Vokabular und ihre Syntax –, war unklar. Die Forscher wiesen auf ein Streudiagramm hin, dessen Cluster schattiert waren, um die semantische Äquivalenz zwischen den Sprachen zu markieren; so sieht die Interlingua von Google Translate offenbar aus. Spätere Versionen der KI-Sprachbehauptung haben sich jedoch auf für Menschen lesbare Beispiele fixiert. Als Facebook 2017 angeblich seinen auf Verhandlungen ausgerichteten Chatbot abschaltete, nachdem dessen Gespräche vom Englischen in knappe, unsinnige Äußerungen abgedriftet waren, zeichneten zahlreiche Medien diese Dialoge wortwörtlich auf. „So sieht eine wirklich nichtmenschliche Sprache aus“, versprach etwa *The Atlantic*, und zitierte: „you i i i everything else... ..“⁴ Kürzlich verkündeten Informatiker der University of Texas in Austin in einem viralen Twitter-Thread, dass DALL-E 2, der

² Devin Coldewey, „Google’s AI Translation Tool Seems to Have Invented Its Own Secret Internal Language“, in: *TechCrunch*, November 2016, <https://techcrunch.com/2016/11/22/googles-ai-translation-tool-seems-to-have-invented-its-own-secret-internal-language/> [Letzter Zugriff: 17.7.2023].

³ Matt Burgess, „Google’s AI Just Created Its Own Universal ‚Language‘“, in: *Wired*, November 2016, <https://www.wired.co.uk/article/google-ai-language-create> [Letzter Zugriff: 17.7.2023].

⁴ Adrienne LaFrance, „What An AI’s Non-Human Language Actually Looks Like“, in: *The Atlantic*, Juni 2017, <https://www.theatlantic.com/technology/archive/2017/06/what-an-ais-non-human-language-actually-looks-like/530934/> [Letzter Zugriff: 17.7.2023].

beliebte Text-zu-Bild-Generator von OpenAI, ebenfalls eine „Geheimsprache“ habe.⁵ Sie kamen darauf, indem sie den Text, der auf generierten Bildern des Modells erschienen war, als neue Prompts in DALL·E 2 einspeisten. „Apoploe vesrreaitais“ bedeutet demnach ‚Vögel‘. „Contarra cctetnxniam luryca tanniounons“ bezeichnet ‚Käfer‘ oder ‚Ungeziefer‘. Die von DALL·E 2 als Antwort auf diese Sätze erzeugten Bilder, die Vögel auf Zweigen oder Essen auf Steinguttellern (für „vicoo-tes“) zeigen, sollten die Behauptungen der Wissenschaftler untermauern.

Die Relevanz solcher Behauptungen für das Lesen von Quellcode liegt auf der Hand, schließlich lässt sich hier leicht eine Art kryptoanalytisches Leitmotiv erkennen. Die ‚geheime‘ oder ‚verborgene‘ KI-Sprache ist wie ein Code geknackt worden – und damit ist vielleicht auch der Vorstoß ins seltsame Terrain nichtmenschlicher Intelligenz gelungen. Außerhalb des populären Diskurses über KI liegt darin sicher ein Körnchen Wahrheit. Die Forschung im Bereich Systemsicherheit und Modellinterpretierbarkeit hat die Auswirkungen beschrieben, die adversariale „Trigger“ auf LLMs haben, etwa Sequenzen von Nonsense-Text wie „zoning tapping fiennes“ und „b 617 matrices dhabi ein wm“.⁶ Wenn sie dem Modellinput angehängt werden, lassen solche Auslöser die LLMs fehlerhaft werden. Manche Sequenzen führen zur völligen Umkehrung von Stimmungsbeurteilungen; andere zwingen Leseverstehenssysteme dazu, immer

⁵ Giannis Daras, „DALL E-2 Has a Secret Language“, Mai 2022, https://twitter.com/giannis_daras/status/1531693093040230402 [Letzter Zugriff: 17.7.2023]; Giannis Daras und Alexandros G. Dimakis, „Discovering the Hidden Vocabulary of DALL E-2“, in: *arXiv*, Juni 2022, DOI:10.48550/ARXIV.2206.00169.

⁶ Eric Wallace u.a., „Universal Adversarial Triggers for Attacking and Analyzing NLP“, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* 10 (2019), S. 2153–2162; Yaman Kmar Singla u.a., „MINIMAL: Mining Models for Universal Adversarial Triggers“, in: *Proceedings of the AAAI Conference on Artificial Intelligence* 36:10 (2022), S. 11330–11339; siehe auch Raphaël Millière, „Adversarial Attacks on Image Generation With Made-Up Words“, in: *arXiv*, August 2022, DOI: 10.48550/arXiv.2208.04135.

wieder dieselbe Antwort zu geben; eine dritte Gruppe stört Entailment-Modelle und eine vierte veranlasst Textgeneratoren dazu, Hassrede wiederzugeben. Selbst hier, außerhalb des Hypes um angebliche KI-Sprachen, ist die Versuchung groß, von ‚triggerbasiertem Codebreaking‘ zu sprechen. Es ist, als ob diese Sequenzen Beweise für einen obskuren Quellcode innerhalb von LLMs wären: Sie scheinen auf eine tiefere programmatische Logik hinzudeuten, die die Modelle antreibt und die mit nur wenigen Zeichen geknackt werden kann.

Über diese Thematik hinaus kann die tatsächliche Praxis der Quellcodekritik, wie sie in der Arbeit von Mark C. Marino sowie Winnie Soon und Geoff Cox veranschaulicht wird, viel dazu beitragen, den linguistischen Status sowohl von Auslösern als auch von Sequenzen wie „vicootes“ zu bewerten.⁷ Denn LLMs lernen nicht von Wörtern. Sie lernen von *Tokens*. Und die ‚Tokenisierungs‘-Algorithmen, die Wörter in Tokens umwandeln, sind für die Leistung des Modells ebenso wichtig wie die Systemarchitektur und die Trainingsdaten. Dies gilt insbesondere für sogenannte ‚Subword‘-Tokens. Quasi-morphematische Einheiten wie „##riated“, „wa“ und „##は“ sind der Stoff, aus dem LLMs geschneidert sind; aber anstelle von Sprache wäre es besser, diese Sequenzen als *Textualität* zu verstehen.⁸ Das ist jedenfalls meine Entgegnung auf die Behauptung der Existenz einer KI-Sprache. Während Linguist:innen und KI-Ethiker:innen die Idee von KI-Sprache über den Begriff der „kommunikativen Absicht“ kritisiert haben,⁹ kann doch das

⁷ Siehe Mark C. Marino, *Critical Code Studies*, Cambridge, MA: The MIT Press 2020, und sein ‚Manifest‘ in diesem Band, S. 27–60; Winnie Soon und Geoff Cox, *Aesthetic Programming: A Handbook of Software Studies*, London: Open Humanities Press 2021.

⁸ Wenn nicht anders angegeben, verwende ich Token aus dem kleingeschriebenen BERT-Modell (Bidirectional Encoder Representations from Transformers) von Hugging Face; Jacob Devlin u.a., „BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding“, in: *arXiv*, Mai 2019, DOI: <https://doi.org/10.48550/arXiv.1810.04805>.

⁹ Emily M. Bender und Alexander Koller, „Climbing Towards NLU: On Meaning, Form, and Understanding in the Age of Data“, in: *Proceedings of the 58th Annual Meeting*

Vorhandensein oder Fehlen solcher Absicht in LLMs nicht die volle Reichweite ihrer materiell-semiotischen Operationen erklären. Wie die neuesten Tokenisierungsalgorithmen besonders deutlich machen, sind LLMs Mediensysteme, die zu Bedeutungen jenseits von Intention fähig sind;¹⁰ Quellcodekritik gibt uns einen Rahmen, um sie als ebensolche Systeme zu betrachten.

Mein Vorschlag, diesen Ansatz zu verfolgen, folgt Michael Gavins Arbeiten über die einzigartigen textuellen Objekte der Sprachmodellierung.¹¹ Für Gavin katalysieren selbst die einfachsten Verfahren der Mathematisierung von Text – wie die Erstellung von Worthäufigkeitslisten oder das Auffüllen von Korpusdokumenten mit Nullen – eine Verschiebung hin zu einer neuen Textualitätsformen, auch wenn es eine Textualität ist, die nicht ganz mit der Bedeutung des Begriffs zusammenfällt, wie die Literaturwissenschaft ihn vielleicht versteht. Die Tokenisierung von Subwörtern ist ein Verfahren jener Art, die Gavin beschreibt, und der Prozess selbst bietet einen äußerst effektiven Blickwinkel, von dem aus sich die „verkettete“ Textualität von LLMs untersuchen lässt. Die Aufgabe dieses Essays besteht darin, dies anhand einer Methode zur Erzeugung von Subwörtern anschaulich zu machen: jenem Algorithmus, den die Google-Forscher zum Aufbau ihres hochgradig verallgemeinerbaren Übersetzungsmodells verwendet haben.

*

of the Association for Computational Linguistics (2020), S. 5185–5198; Emily M. Bender u.a., „Über die Gefahren stochastischer Papageien. Können Sprachmodelle zu groß sein?“, in diesem Band, S. 277–331.

¹⁰ Siehe Lydia H. Liu, *The Freudian Robot: Digital Media and the Future of the Unconscious*, Chicago: University of Chicago Press 2010; Maurizio Lazzarato, *Signs and Machines: Capitalism and the Production of Subjectivity*, Los Angeles: Semiotext(e) 2014, sowie Hannes Bajohr, „Dumme Bedeutung. Künstliche Intelligenz und Artifizielle Semantik“, in diesem Band, S. 197–215.

¹¹ Michael Gavin, „Is There a Text in My Data? (Part 1): On Counting Words“, in: *Journal of Cultural Analytics* 5:1 (2019), S. 1–24.

Unter Tokenisierung versteht man das Zerlegen von Fließtext in zählbare Einheiten. Diese Einheiten oder Tokens sind für die Verarbeitung natürlicher Sprache (*natural language processing*, NLP) von zentraler Bedeutung, denn sie sind das, was in der Arithmetik der stochastischen Semantik buchstäblich zählt. Ihre Verwendung ist seit den frühesten Sprachmodellen nahezu konstant und geht auf die Arbeiten von Andrei Markov sowie auf Claude Shannons *Mathematische Theorie der Kommunikation* zurück. Um sich einer Sprache über statistische Stichproben zu nähern, verwendete Shannon meist zeichenbasierte Tokens von unterschiedlicher Länge.¹² Token mit zwei Zeichen, sogenannte Bigramme, dienten als Grundlage für seine ersten Modelle; Modelle, die er aus Trigrammen – Token mit jeweils drei Zeichen – erstellte, „kamen einer Sprache“ jedoch „näher“: „IN NO IST LAT WHEY“, heißt es da, und: „CRATIC FROURE BIRS GROCID PONDENOME“. Wenn das eine Verbesserung darstellt, dann nicht, weil Shannon seine Modelle mit Blick auf linguistische Prinzipien verfeinert hatte. Zentral ist, dass er, wie viele andere Informatiker:innen auch, seine Tokenisierungsmethoden ad hoc entwickelte und sie spontan an ein sich veränderndes Problemfeld anpasste.¹³ Meistens ist ein Token einfach ein Wort – wenn man unter ‚Wort‘ eine beliebige Folge von Zeichen zwischen zwei Leerzeichen versteht.

Seit den 1950er-Jahren war diese Definition von Token vorherrschend, obwohl in den späten 1980er und frühen 90er-Jahren gelegentlich von der Tokenisierung mit Leerzeichen abgewichen wurde. Forscher:innen im Bereich des Information Retrieval experimentierten mit „sublexikalischen“ Tokens, die Wörter aus kleineren, kombinatorischen Einheiten zusammensetzten, um den notwendigen Wortschatz ihrer Suchsysteme

¹² Claude Elwood Shannon und Warren Weaver, *The Mathematical Theory of Communication*, Urbana: University of Illinois Press 1998, S. 43.

¹³ Siehe David Golumbia, *The Cultural Logic of Computation*, Cambridge, MA: Harvard University Press 2009.

zu verkleinern.¹⁴ Diese verstreuten Experimente scheinen seinerzeit keinen großen Einfluss gehabt zu haben, doch tauchte die Idee in den 2010er-Jahren wieder als zentrale Referenz auf, als sich die Forschungsbemühungen von Google und Facebook auf das Problem unbekannter Daten, sogenannter ‚Out-of-Vocabulary‘-Elemente (OOV) konzentrierten.¹⁵ Wie die Ingenieur:innen der ersten Welle maschineller Übersetzung feststellen mussten, sind Modelle nur in der Lage, Token zu verarbeiten, die in ihren ursprünglichen Trainingsvokabularen enthalten sind. Das heißt, seltene oder nicht vorhandene Wörter sind im realweltlichen Einsatz besonders problematisch. Techniken für den Umgang mit OOV-Elementen gibt es viele, aber bei Google und Facebook konzentrierte sich der Ansatz darauf, die statische Natur von Trainingsvokabularen zu überdenken. Die Forschenden stellten die Theorie auf, dass Modelle besser mit unbekanntem Daten umgehen können, wenn sie beim Training mit den atomaren Bausteinen von Wörtern ausgestattet werden, die je nach Bedarf wieder zusammengesetzt oder neu kombiniert werden können. Mit anderen Worten: Die Modelle mussten ‚lernen‘, wie man buchstabiert.

¹⁴ Siehe Roy Kimbrell, „Searching for Text? Send an N-Gram!“, in: *Byte* 13:5 (1988), S. 297–312; Hinrich Schütze, „Word Space“, in: *Advances in Neural Information Processing Systems* 5 (1992), S. 895–902.

¹⁵ Tomáš Mikolov u.a., „Subword Language Modeling with Neural Networks“, 2011, S. 4; Mike Schuster und Kaisuke Nakajima, „Japanese and Korean Voice Search“, in: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing* (2012), S. 5149–5152; Piotr Bojanowski u.a., „Enriching Word Vectors with Subword Information“, in: *arXiv*, Juni 2017, DOI: <https://doi.org/10.48550/arXiv.1607.04606>.



Abb. 1:
Suche mit sublexikalischen Token, Illustration aus der Zeitschrift *Byte*¹⁶

Um sie darauf zu trainieren, galt es, die Tokenisierungsoperation enger an die Funktionsweise der LLMs zu binden. Im Gegensatz zu anderen Methoden der Vorbereitung von Text für die Sprachmodellierung verfügen Modelle wie BERT oder GPT-3 über eingebaute Tokenisierer, die die Rohdaten während des Trainings und späterer nachgelagerter Aufgaben automatisch verarbeiten; die Modellentwickler:innen müssen nur wenige Anweisungen für den Ablauf dieses Prozesses machen, einmal abgesehen von der Angabe einer Zielgröße für die resultierenden Vokabulare (so gewünscht). Der Rest bleibt in der Blackbox. Das Ergebnis: Modelle, die Witze erklären oder

¹⁶ Kimbrell, „Searching for Text? Send an N-Gram!“.

ausführliche Antworten auf Prompts geben können, und Subwörter, die „pre“, „##ters“, „##b“ und „☆“ lauten.

Viele NLP-Bibliotheken verwenden für ihre Tokenizer den sogenannten „Byte-Pair-Encoding“-Algorithmus (BPE), der für das Google-Translate-Modell von 2016 entwickelt wurde – jenes Modell, welches schließlich wegen seiner angeblichen Interlingua die Aufmerksamkeit der Medien auf sich zog.¹⁷ Der Algorithmus, den ich unten in Python wiedergebe, ist bemerkenswert einfach. Er umfasst nur zwei Schritte. Zunächst werden in einem Korpus die Bigramme der Zeichen gezählt. Dann werden diese Bigramme auf der Grundlage dieser Zählungen schrittweise zu längeren Sequenzen verkettet, um die Ausgabe den ursprünglichen Daten anzunähern.

Schritt 1

```

01 # Die Tokenisierung beginnt mit einem vorgenerierten Wörterbuch
02 # aus Wortzählungen. Die Zählungen werden aus einem Durchlauf durch
03 # einen Leerzeichen-Tokenizer generiert, der zudem Leerzeichen
04 # zwischen jedes Zeichen in jedem Wort einfügt. In einem weiteren
05 # Vorbereitungsschritt werden Korpusdokumente mit speziellen Tags
06 # versehen, um Sequenzgrenzen zu markieren; das Wörterbuch
07 # berücksichtigt auch diese Tags. BERT verwendet [CLS] und [SEP],
08 # um den Anfang bzw. das Ende längerer Sequenzen wie Sätzen zu
09 # markieren; ein anderer Tokenizer verwendet Tilden oder Unterstriche,
10 # um führende/nachlaufende Leerzeichen darzustellen; die Version von
11 # 2016 markiert nur die Enden von Wörtern mit </w>.
12 vocab = {
13     "l o w </w>": 5,
14     "l o w e r </w>" 2,
```

¹⁷ Rico Sennrich, Barry Haddow und Alexandra Birch, „Neural Machine Translation of Rare Words with Subword Units“, in: *arXiv*, Juni 2016, DOI: <https://doi.org/10.48550/arXiv.1508.07909>.

```

15     "newest </w>": 6,
16     "widest </w>": 3
17 }
18
19 import re
20 import collections
21
22 # Bei der Initialisierung ruft der Tokenizer eine Funktion auf,
23 # die die Zählungsdaten lädt ...
24 def get_stats(vocab):
25     # ... und ein neues, leeres Wörterbuch erstellt.
26     pairs = collections.defaultdict(int)
27
28     # Dann werden alle Wörter in der Eingabe durchlaufen und
29     # jedes in eine Liste von Zeichen zerlegt.
30     for word, freq in vocab.items():
31         symbols = word.split()
32
33         # Für jedes dieser aufgespaltenen Wörter erzeugt die Funktion
34         # eine Menge aufeinanderfolgender Bigramme und fügt sie ihrem neuen
35         # Wörterbuch hinzu. Sie gibt jedem die entsprechende Anzahl des
36         # Wortes, von dem diese Bigramme abgeleitet sind. Wenn ein Bigramm
37         # bereits im Wörterbuch ist, weil es von einem vorherigen Wort
38         # abgeleitet wurde, aktualisiert die Funktion die Anzahl dieses
39         # Bigramms, indem sie die neue Anzahl zur alten hinzufügt.
40         for i in range(len(symbols) - 1):
41             pairs[symbols[i], symbols[i + 1]] += 1
42
43     # Wenn die Funktion alle Wörter in ihrer Eingabe durchlaufen hat,
44     # gibt sie das neue Wörterbuch zurück.
45     return pairs

```

Schritt 2

```

43 # Mit den neu generierten Zählungen wählt der Tokenizer
44 # ein Bigramm aus. Dieses Bigramm ist entweder das
45 # häufigste in den Zählungen oder dasjenige, das die
46 # Gesamtwahrscheinlichkeit eines einfachen,
47 # probabilistischen Sprachmodells, das an das Korpus
48 # angepasst ist, am meisten erhöht. Unabhängig von der
49 # Metrik ruft der Tokenizer nach der Auswahl eines Bigramms
50 # eine zweite Funktion auf, die das ausgewählte Bigramm
51 # und das ursprüngliche Wörterbuch der geteilten Wörter übergibt.
52 def merge_vocab(pair, v_in):
53     # Die Funktion erstellt ein neues Wörterbuch ...
54     v_out = {}
55
56     # ... und verwandelt das Bigramm in eine durchsuchbare Zeichenkette.
57     bigram = re.escape(' '.join(pair))
58     p = re.compile(r'(?!\S)' + bigram + r'(?!\S)')
59
60     # Anschließend wird im Eingabewörterbuch nach
61     # diesem Bigramm gesucht.
62     for word in v_in:
63         # Wenn sie eine Übereinstimmung in der Eingabe
64         # findet, verkettet die Funktion die linken
65         # und rechten Komponenten dieses Bigramms zu
66         # einem einzigen Token. Zum Beispiel wird das
67         # Eingabewort "l o w e r </w>" mit dem Bigramm
68         # "er" zu "l o w er </w>" verkettet. Wenn das
69         # verkettete Bigramm in der Mitte eines Wortes
70         # steht, fügen einige Versionen des
71         # Tokenisierungsalgorithmus, wie der von BERT
72         # verwendete, ein neues Tag hinzu, um dies zu
73         # markieren ("l o w ##er </w>").
74     w_out = p.sub(' '.join(pair), word)

```

```

75
76     # Das so hergestellte Bigramm wird in das neue
77     # Wörterbuch aufgenommen und ihm wird die Häufigkeitsanzahl
78     # des Eingabeworts zugeordnet.
79     v_out[w_out] = v_in[word]
80
81     # Sobald die Funktion alle möglichen Bigramm-Übereinstimmungen
82     # in ihrer Eingabe gefunden hat, gibt sie das neue Wörterbuch
83     # zurück. Schritt zwei ist damit abgeschlossen.
84     return v_out
85
86 # Ein einziger Durchlauf durch diese beiden Schritte bringt
87 # nicht viel, aber der Tokenizer wird diesen Prozess mehrere
88 # (oft sehr viele) Male wiederholen.
89 num_merges = 5
90
91 # Nachdem er alle Instanzen eines ausgewählten Bigramms
92 # gefunden und verkettet hat, kehrt er zum ersten Schritt
93 # zurück und zählt jedes Bigramm aus der neuen, modifizierten
94 # Version der Wortzählungen.
95 for _ in range(num_merges):
96     # Diesmal spiegeln die resultierenden Zählungen die
97     # Sequenzen wider, die die neu gebildeten Tokens enthalten,
98     # und nicht jedes ihrer einzelnen Zeichen.
99     pairs = get_stats(vocab)
100
101     # Diese neuen Zählungen werden dann beeinflussen,
102     # welche Bigramm-Komponenten als Nächstes ausgewählt
103     # und verkettet werden sollen ...
104     best = max(pairs, key = pairs.get)
105
106     # ... was wiederum die nachfolgenden Zählungen und
107     # die Auswahl weiterer Bigramme beeinflusst und
108     # so weiter. Allmählich werden sich aus den

```

```

109 # Zeichenfolgen erkennbare Wörter herauschälen.
110 # Ein paar Hundert Iterationen nach der Zusammenführung
111 # von "er" könnte der Tokenizer "ow" auswählen,
112 # um "l ow er </w>" sowie "p i l l ow </w>", "ow </w>",
113 # "ow n er </w>" und mehr zu bilden. Der Prozess
114 # wird fortgesetzt, wobei Sequenzen innerhalb von Wörtern
115 # zusammengeführt werden, bis eine von zwei Bedingungen
116 # erfüllt ist: Entweder gehen dem Tokenizer die Bigramme
117 # aus, die er erstellen kann (in diesem Fall spiegelt
118 # seine Ausgabe seine Eingabe wider), oder, was häufiger
119 # der Fall ist, er erreicht eine vorgegebene Anzahl von
120 # Subwörtern; alle verbleibenden Bigramme werden
121 # unverändert belassen. Wenn eine der beiden Bedingungen
122 # erfüllt ist, ist der Prozess beendet und die
123 # tokenisierten Korpusdaten sind bereit für die Modellierung.
124 # Einige Wörter werden neu zusammengesetzt, andere nicht,
125 # aber in beiden Fällen wird ein Modell in der Lage sein,
126 # sie während des Trainings und nachgelagerter Aufgaben
127 # darzustellen.
128 vocab = merge_vocab(best, vocab)

```

Original

low </w>
lower </w>
newest </w>
widest </w>

Zusammengeführt

low </w>
low er </w>
new est</w>
wi d est</w>

Ergebnis von fünf Iterationen durch den BPE-Tokenizer 2016

Zusammengenommen sind die oben genannten Code-Blöcke eine eher langweilige Antwort auf die so erstaunliche These einer KI-Sprache. Dass Modelle wie DALL·E 2 auf Sequenzen wie „Apoploe vesrreaitais“ reagieren, hat weit weniger mit Erfindung als mit Verkettung zu tun, einem Prozess, bei dem LLMs unscharfe, ‚gebootstrapte‘ Repräsentationen ihrer Trainingsdaten aneinanderreihen, „ohne sich auf die Semantik zu konzentrieren.“¹⁸ Noch einmal: Wie lernt man, ein Wort zu benutzen, das man noch nie gesehen hat? Man fängt damit an, es auszubuchstabieren: „A pop loe v es r re ait ais“.¹⁹ Aber Sequenzen mit fester Länge, wie sie Shannon modelliert hat, bieten dafür kein geeignetes Material. Der Schlüssel zu den Veränderungen, die Subwörter mit sich bringen, ist Variabilität. Sie sind das Ergebnis einer alles durchdringenden Gleichgültigkeit gegenüber der Frage, was die atomaren Elemente von Textdaten sind. Subwörter können einzelne Zeichen aus verschiedenen Schriftsystemen sein: präfixähnliche Bigramme, Brocken, die isomorph wie Morpheme aussehen, oder sogar – und das ist am merkwürdigsten – ganze Wörter. Unabhängig von ihrer Form wird ein Modell für jede solche Sequenz ein *Embedding* erzeugen. Subwörter haben in diesem Sinne einen tiefgreifenden Verflachungseffekt auf den Text. Mit ihnen unterscheidet sich ein Wort in seinem Wesen nicht von einem einzelnen Zeichen.

*

Während eine solche Verflachung den Grad, in dem Subwörter als Sprache zählen können, beeinträchtigt, bleibt ungeklärt, ob

¹⁸ Schuster und Nakajima, „Japanese and Korean Voice Search“, S. 5150.

¹⁹ Diese Tokens stammen aus dem OpenAI-Tokenizer-Tool, das eine Version des GPT-3-Tokenizers verwendet, der von Nick Walton implementiert wurde, „GPT-3-Encoder“, 2020, <https://www.npmjs.com/package/gpt-3-encoder> [Letzter Zugriff: 17.7.2023]; siehe auch: <https://beta.openai.com/tokenizer> [Letzter Zugriff: 17.7.2023].

und wie sie zu lesen sind. Auch hier gilt mein Interesse der einzigartigen Textualität von LLMs und ihren Subwörtern. Wenn wir es hier tatsächlich mit einer Veränderung oder gar Infragestellung der Art und Weise zu tun haben, wie man solche Subwörter liest, muss der Grenzfall für diesen Unterschied dort zu suchen sein, wo eine tokenisierte Zeichenkette mit einem Wort zusammenfällt. Was ist, anders gefragt, der Unterschied zwischen dem Wort „lower“ und dem Subwort „lower“? In gewissem Sinne gibt es keinen – und dennoch: Der Unterschied zwischen diesen Zeichenfolgen ist, wie Marcel Duchamp sagen würde, infradünn (*inframince*) – aber ein Unterschied ist da eben vielleicht doch.

Ich werde mich diesem Unterschied auf zwei etwas verkürzten Pfaden nähern. Der erste ist der der Orthografie. Die Intuition, dass man sich Subwörter in Bezug auf Rechtschreibung oder Zeichensetzung als Analoga von Wörtern vorstellen kann, geht nicht ganz fehl; die verschiedenen Konventionen, die Typograf:innen entwickelt haben, um Wörter in einem Textblock zu unterteilen, sind weitere Vergleichspunkte.²⁰ Aber es gibt auch die spezielle Orthografie dessen, was Pip Thornton „Sprache im Zeitalter algorithmischer Reproduktion“ nennt.²¹ Von „program, ##matic, token, ##ization“ lässt sich eine direkte Verbindung zur allgemeineren textuellen Bedingung des linguistischen Kapitalismus ziehen, in der Dienste wie Autovervollständigung das Schreiben in vorhersehbare (sprich: wirtschaftlich verwertbare) Ausdrücke zwingen.²² Die Kund:innen von Google Ads, schreibt Frédéric Kaplan, geben „nicht viele

²⁰ Im englischen Sprachkontext formuliert einer der bekanntesten Stilratgeber, *Hart's Rules*, Worttrennungen in vorausweisender Funktion: „Das Prinzip besteht darin, dass der Teil des Wortes, der am Ende einer Zeile steht, auf den Teil hinweisen soll, mit dem die nächste Zeile beginnt“. (*Hart's Rules for Compositors and Readers at the University Press*, Oxford, London: Oxford University Press 1921, S. 54.)

²¹ Pip Thornton, *Language in the Age of Algorithmic Reproduction: A Critique of Linguistic Capitalism*, Dissertation, University of London 2018.

²² Siehe John Cayley, „Terms of Reference and Vectoralist Transgressions“, in: *Amo- dern*, Oktober 2013, <https://amodern.net/article/terms-of-reference-vectoralist-trans->

Gebote für falsch geschriebene Wörter ab“ – ein Punkt, den Jeff Dean, der Leiter von Google AI, einräumen müsste.²³ Orthografische Normalisierung ist das Geschäftsmodell des Unternehmens seit seinen Anfängen. Dean berichtet: „Im Jahr 2001 erkannten einige Kolleg:innen, die bei Google nur ein paar Meter von mir entfernt saßen, dass sie eine obskure Technik namens maschinelles Lernen einsetzen konnten, um falsch geschriebene Suchanfragen zu korrigieren.“²⁴ Viele der heutigen LLMs – und die für sie erforderlichen Subwörter – gehen auf diesen Moment zurück.

Anders ausgedrückt: In der Orthografie des Sprachkapitalismus ist ein Token (Texteinheit) immer auch ein Token (Währungseinheit). Thorntons Projekt „{poem}.py“, das die Kosten kanonischer Gedichte entsprechend den Google-AdWords-Preisen ihrer Inhalte auflistet, macht dies besonders deutlich.²⁵ Aber wenn man hinter den Thermoausdruck für Wordsworths „Daffodils“ schaut, findet man auch eine ganze Reihe brandneuer Modellierungsinfrastrukturen, die im Zuge der weitverbreiteten Einführung von LLMs entstanden sind. Man denke an OpenAI, das seine Preise direkt über Subwort-Tokens festlegt. Da der Quellcode für die neuesten Modelle der GPT-Serie unter Verschluss gehalten wird, sind die Dienste des Unternehmens stattdessen über eine kostenpflichtige API zugänglich. Die Preise für die Verwendung der Basismodelle liegen zwischen 0,0008 und 0,06 \$ pro Stapel von 1.000 Tokens; benutzer:innendefinierte Modelle, die auf einem Datensatz Ihrer Wahl trainiert werden, kosten bis zu 0,12 \$ pro Stapel. In

gressions/ [Letzter Zugriff: 17.7.2023]; Frédéric Kaplan, „Linguistic Capitalism and Algorithmic Mediation“, in: *Representations* 127:1 (2014), S. 57–63.

²³ Ebd., S. 59.

²⁴ Jeff Dean, „Introducing Pathways: A Next-Generation AI Architecture“, in: *The Keyword*, Oktober 2021, <https://blog.google/technology/ai/introducing-pathways-next-generation-ai-architecture/> [Letzter Zugriff: 17.7.2023].

²⁵ Pip Thornton, „{poem}.py“, 2016, <https://culture.theodi.org/poem-py/> [Letzter Zugriff: 17.7.2023].

einem Leitfaden zur Preisgestaltung heißt es: „Sie können sich Token als Teile von Wörtern vorstellen, wobei 1.000 Token etwa 750 Wörtern entsprechen“ – ein praktisches Missverhältnis, denn die Kund:innen werden am Ende mehr für die Modellierungsdienste von OpenAI bezahlen, als ihre Wortzahl vermuten ließe. OpenAI formuliert diesen Haken mit ironischer, Wittgenstein'scher Wörtlichkeit: „Many| words| map| to| one| token|,| but| some| don't|:| ind|iv|isible“.

Zu den Preisschemata von OpenAI könnte man noch Tausende von Jupyter-Notebook-Tutorials, viele der auf Hugging Face Spaces gehosteten Apps, Prompt-Engineering-Services,

GPT-3 Codex

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing the underlying bytes: 🍌

Sequences of characters commonly found next to each other may be grouped together: 1234567890

Clear Show example

Tokens	Characters
64	252

Many words map to one token, but some don't: indivisible.

Unicode characters like emojis may be split into many tokens containing the underlying bytes: 🍌🍌🍌🍌

Sequences of characters commonly found next to each other may be grouped together: 1234567890

TEXT TOKEN IDS

Abb. 2:
OpenAIs Tokenizer-Werkzeug

„Chain Authoring“²⁶ und Start-ups hinzufügen, die sich ausschließlich der Optimierung von Modellen zu ihrem kostengünstigeren Einsatz widmen. Die alltägliche Praxis der Sprachmodellierung ist jetzt in einer ganzen „semiotischen Infrastruktur“ angesiedelt, die sich der Verwaltung und Bearbeitung von Textdaten widmet.²⁷ Ich gehe davon aus, dass die Kartierung der vielen Verkettungen, die diese neue Infrastruktur unterstützen, eine der Hauptaufgaben künftiger Quellcode-Kritiker:innen sein wird. Aber auch die Auswirkungen dieser Verkettungen lassen sich auf der Oberfläche der LLM-Ergebnisse selbst erkennen. Sie bestehen zum Beispiel in dem Unterschied zwischen „indivisible“ und „ind|iv|isible“, und sogar in Momenten, in denen, wie bei „lower“ und „lower“, dieser Unterschied infradünn wird.

*

Mein zweiter Pfad ist semiotischer Natur und erfordert die Erforschung der Ursprünge der Tokenisierung von Subwörtern. In seiner aktuellen Form ist der BPE-Algorithmus ein Produkt des modernen maschinellen Lernens. Seltsamerweise stammt er nicht von früheren Textaufbereitungsmethoden im NLP, wie dem Stemming oder der Lemmatisierung. Seine Entwickler:innen folgten stattdessen einem unter Anwender:innen des maschinellen Lernens inzwischen üblichen Muster, indem sie die Technik aus einem entfernten Forschungsbereich bezogen und ad hoc in ihr eigenes System bei Google einfügten.²⁸ BPE

²⁶ Tongshuang Wu u.a., „PromptChainer: Chaining Large Language Model Prompts Through Visual Programming“, in: *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022.

²⁷ Leif Weatherby und Brian Justie, „Indexical AI“, in: *Critical Inquiry* 48:2 (2022), S. 382.

²⁸ Siehe Adrian Mackenzie, *Machine Learners: Archaeology of a Data Practice*, Cambridge, MA: The MIT Press 2017; Jonathan Roberge und Michael Castelle (Hg.), *The Cultural Life of Machine Learning: An Incursion into Critical AI Studies*, Cham: Palgrave Macmillan 2021.

war ursprünglich als Datenkomprimierungstechnik gedacht und wurde erstmals 1994 vom Softwareingenieur Philip Gage veröffentlicht. Gages Version basiert auf der gleichen Bigramm-Logik wie die heutigen Tokenizer, aber anstatt Textabschnitte abzugleichen und zusammenzuführen, wählt sie Paare benachbarter Bytes in einem Datenblock aus und codiert sie in ein einziges Byte. Anschließend baut es iterativ auf den erstellten Einheiten auf. Bytes, die für Paare stehen, werden in wieder neue Bytes codiert, diese dann wieder in neue Paare und so weiter, bis der Kompressor seine Eingabedaten auf den kleinstmöglichen Fußabdruck reduziert hat.

Gage nennt ein Beispiel: Eine Sequenz wie ABABCABCD würde zu XXCXCD, wobei X für AB steht; die neue Sequenz würde wiederum zu XYXD, wobei Y für XC steht.²⁹ Ebenso wird „l o w e r </w>“ zu „l o w e r </w>“ und schließlich zu „l o w e r </w>“, „l o w e r </w>“, „l o w e r </w>“ und schließlich „l o w e r </w>“. Bezeichnenderweise handelt es sich im letzteren Fall bei den Sequenzen, die ein Tokenizer zur Darstellung von Subwörtern verwendet, nicht um externe Daten, sondern um Fragmente der ursprünglichen Zeichenfolgen im Korpus. In einer Art Umkehrung der ursprünglichen BPE-Logik codiert die Tokenisierung von Subwörtern Texte, indem sie sie in längere Sequenzen dekomprimiert, bevor sie sie wieder in besser erkennbare Zeichenketten decodiert.

Alexandra Schneiders Definition von Kompression als „Reduktion von Daten bis an die Schwelle der Verständlichkeit“, wäre eine Möglichkeit, meine Frage nach dem Grenzfall zwischen Wort und Subwort im Kontext des ursprünglichen BPE-Algorithmus zu formulieren.³⁰ Aber auch Gage selbst

²⁹ Philip Gage, „A New Algorithm for Data Compression“, in: *The C Users Journal* 12:2 (1994), S. 3.

³⁰ Alexandra Schneider, „Viewer's Digest: Small Gauge and Reduction Prints as Liminal Compression Formats“, in: Marek Jancovic, Axel Volmar und dies. (Hg.), *Format Matters: Standards, Practices, and Politics in Media Cultures*, Lüneburg: meson press 2019, S. 140.

bewegte sich 1996 in Richtung Text, als er seinen Kompressor neu schrieb.³¹ Diese zweite Version akzeptiert nur Textdateien. Sie funktioniert ganz ähnlich wie sein Programm von 1994, allerdings fügte Gage einen interessanten Check hinzu, der ausgeführt wird, bevor das Programm mit der Komprimierung der Daten beginnt. Im Folgenden gebe ich ein Fragment des relevanten Quellcodes (in C) wieder:

```

1  /* Der Kompressor erstellt neben anderen Makros eine maximale
2  * Größenbegrenzung für die Datenblöcke, die er einliest. */
3  #define MAXSIZE 65535L
4
5  /* Damit ist eine Funktion definiert, die Argumente für eine
6  * zu komprimierende Datei (Eingabe) und die komprimierte Version
7  * dieser Datei (Ausgabe) akzeptiert. */
8  void compress (FILE *input, FILE *output)
9  {
10     /* Mehrere Variablen werden am Anfang dieser Funktion
11     * initialisiert. Ich werde nur eine von ihnen auflisten:
12     * die Variable, die die Pufferzuweisung verwaltet, die der
13     * Größe des oben formulierten Makros entspricht. Wenn der
14     * Kompressor diese Variable zuweist, wird ein Pointer
15     * auf den Speicherplatz, an dem die Eingabedatei gespeichert
16     * werden soll, eingerichtet. */
17     buffer = (unsigned char *)malloc(MAXSIZE);
18
19     /* Nachdem alle Variablen initialisiert sind, liest die Funktion
20     * die Eingabedatei in den Speicherpuffer und führt die erste
21     * von zwei Fehlerprüfungen durch. */
22     size = fread(buffer,1,MAXSIZE,input);
23

```



³¹ Philip Gage, „Byte Pair Encoding“, September 1997, <https://github.com/Algorithms-in-cpp/CUJ-1990-2000/tree/1fd9fc2ed887229c0c9774e408e2c0f17e010910/SOURCE/1997/SEP97> [Letzter Zugriff: 17.7.2023].

```

24      /* Zunächst wird festgestellt, ob die Datei zu groß zum
25      * Komprimieren ist. Falls ja, gibt der Kompressor eine
26      * Fehlermeldung an den:die Benutzer:in aus und bricht ab. */
27      if (size == MAXSIZE) {
28          printf("File too big\n");
29          exit(1);
30      }
31
32      /* Wenn die Datei die Größenprüfung besteht, führt der
33      * Kompressor eine zweite Prüfung durch. Er durchläuft jedes
34      * Byte im Puffer und ... */
35      for (i=0; i<size; i++)
36          /* ... für jedes Byte prüft er, ob die Zahl, die dieses
37          * Byte repräsentiert, größer als 127 ist. */
38          if (buffer[i] > 127) {
39              /* Wenn das Byte größer als 127 ist, bricht der
40              * Kompressor ebenfalls ab, wieder mit einer Fehlermeldung.
41              * Der Grund, warum er nichts akzeptieren kann, was
42              * größer als diese Zahl ist, hat mit einer Entscheidung
43              * von Gage zu tun: Im Gegensatz zum ursprünglichen
44              * Kompressor, der nach jedem unbenutzten Byte im
45              * Speicherblock sucht, reserviert die Textversion alle
46              * höherwertigen Bytes (128-256) für die Codierung von
47              Paaren. */
48              printf("Dieses Programm funktioniert nur mit Textdateien\n");
49              exit(1);
50      }

```

Im Text stellen die höherwertigen Bytes, auf die der Gage-Kompressor prüft, den erweiterten Satz von ASCII-Zeichen dar. Die praktische Konsequenz dieser Prüfung ist, dass jedes Zeichen innerhalb dieses Satzes vom Algorithmus nicht komprimiert werden kann. Zeichen mit diakritischen Zeichen (Á, ñ),

bestimmte mathematische und Währungssymbole (\pm , $\frac{1}{4}$, ¥), Zeichen für die Darstellung von einfachen Textgrafiken (, ) und andere führen zu einem Fehler: „Dieses Programm funktioniert nur mit Textdateien.“ Anders ausgedrückt: Alles, was über das hinausgeht, was das Unicode-Konsortium als ‚Basic Latin‘ bezeichnet, ist in diesem Quellcode *kein Text* oder *Text*.

Es gäbe viel darüber zu sagen, wie diese Auslöschung die sprachliche Hegemonie des lateinischen Alphabets wieder in digitale Medien einschreibt; unter anderem wäre Lydia Lius Arbeit über Shannon und „Printed English“ ein wichtiger Bezugspunkt. Vorerst werde ich dies jedoch nur als zukünftigen Ort von Kritik erwähnen – und als Vorstufe daran erinnern, dass die BPE-Tokenisierung für heutige LLMs oft dazu dient, genau die Art von Text handzuhaben, die Gage auslöscht. Dies verleiht der Verbreitung von BPE eine ironische Dimension. Aber um zum Schluss zu kommen, möchte ich noch eine andere Dimension der Aussage „Dieses Programm funktioniert nur mit Textdateien“ verfolgen. Denn sie legt auch nahe, dass die Ausgabe der BPE-Kompression selbst Text ist. Das heißt, wenn ASCII höherer Ordnung Text ist, dann fallen alle Daten, die in diesem Bereich codiert werden, in dieselbe Kategorie einschließlich der Subwörter. Die Aussage in diesem zweiten Sinne zu lesen, bedeutet daher, zu den Überlegungen zurückzukehren, die ich bereits angestellt habe: Was ist dieser Text und in welcher Weise könnte er die einzigartige Textualität von Subwörtern prägen – diesen Tokens, die weder Sprache noch Wörter sind?

In den Quellcode-Dateien der aktuellen BPE-Tokenizer gibt es eine Namenskonvention, die eine Antwort nahelegt. Die Code-Blöcke für das Google Translate-Modell von 2016 haben davon bereits Gebrauch gemacht:

```
28 for word, freq in vocab.items():
29     symbols = word.split()
```

Hier ist es (in Rust) im BERT-Tokenizer, der bei Hugging Face gehostet wird:³²

```

1  #[derive(Debug, Clone, Copy)]
2  struct Symbol {
3      c: u32,
4      prev: usize,
5      next: usize,
6      len: usize
7  }
8  impl Symbol {
9      /// Verschmilzt das aktuelle Symbol mit dem nächsten.
10     /// Um prev/next zu aktualisieren, betrachten
11     /// wir Self als das Symbol auf der linken Seite
12     /// und other als das auf der rechten Seite.
13     pub fn merge_with(&mut self, other: &Self, new_c: u32) {
14         self.c = new_c;
15         self.len += other.len;
16         self.next = other.next;
17     }
18 }

```

An anderer Stelle erscheint die Konvention in erklärenden Kommentaren, wie in der unten stehendes Python-Funktion, die Teil des GPT-2-Encoders von OpenAI ist:³³

```

1  def get_pairs(word):
2      """Gibt eine Menge von Symbolpaaren in einem
3      Wort zurück. Das Wort wird als ein Tupel von
4      Symbolen dargestellt (Symbole sind Zeichenketten

```

³² Hugging Face (Hg.), „Tokenizers“, <https://github.com/huggingface/tokenizers/releases/tag/v0.13.1> [Letzter Zugriff: 17.7.2023].

³³ Alec Radford u.a., „Language Models Are Unsupervised Multitask Learners“, 2019, <https://github.com/openai/gpt-2> [Letzter Zugriff: 17.7.2023].

```

5     variabler Länge).
      """
6     pairs = set()
7     prev_char = word[0]
8     for char in word[1:]:
9         pairs.add((prev_char, char))
10        prev_char = char
11    return pairs

```

Für diese und andere Tokenisierer sind die durch die Tokenisierung von Subwörtern erzeugten verketteten Zeichenketten *Symbole*. Trotz der vielen jüngsten Behauptungen, dass Deep Learning einen Paradigmenwechsel in der Praxis und Erkenntnistheorie der KI herbeigeführt hat, bleibt die zeitgenössische Sprachmodellierung in einem wichtigen Sinne an den Bereich des Symbolischen gebunden. In diesem Bereich steht ein Symbol für eine Verknüpfung, die wiederum für eine andere stehen kann. Ganze Ketten von Verknüpfungen werden so bis zur Schwelle der Verständlichkeit zu abgeflachten Zeichenfolgen variabler Länge verdichtet. Eine solche Kette verknüpft „lower“ mit seiner infradünnen Variante, „lower“. Andere Verkettungen gibt es zuhauf. Alle markieren einen Platz, notieren.

Dabei stellen sie, wie ich meine, die Grenzen dessen, was Modellierung ausmacht, infrage. Denn mit Subwörtern modellieren LLMs nicht Sprache, auch nicht Tokens oder Text. Sie modellieren **Text**, der selbst ein Bootstrap-Modell der Korpusdaten ist, aus denen er abgeleitet wurde. Große Sprachmodelle modellieren Modelle. Und uns bleibt nichts anderes zu tun, als ihre Ergebnisse zu lesen.

Anhang:

Im Folgenden habe ich die Ergebnisse von mehreren Hundert Iterationen des BPE-Tokenizers 2016 dokumentiert. Das Korpus ist dasselbe, den die Google-Forscher in ihrer ursprünglichen Version des Algorithmus verwendet haben, und enthält Anzeigen für Bauunternehmen und Reisebüros, Auszüge aus dem PHP-Handbuch, Lizenzierungsklauseln, Geschäftsbedingungen und Bibelverse – mit anderen Worten: typischen Internetttext.

1 Iteration

iron cement is a ready for use pastewhich
is laid as a fillet by putty knife or finger
in the mould edges (corners) of the st
eel ingot mould . iron cement protects t
he ingot against the hot , abrasivesteel
casting process . a fire resistant repair c
ement for fireplaces , ovens , open fire
places etc . construction and repair of
highways and . . . an announcement must be
commercial character . goods and servic
es advancement through the P.O.Box syste
m is NOT ALLOWED . deliveries (spam) an
d other improper information deleted . t
ranslator Internet is a Toolbar for MS I
nternet Explorer . it allows you to tran
slate in real time any web page from one l
anguage to another .

250 Iterationen

iron cement is a ready for use paste which is laid as a fillet by putty knife or finger in the mould edges (corners) of the steel ingot mould. iron cement protects the ingot against the hot, abrasive steel casting process. a fire resistant repair cement for fire places, ovens, open fireplaces etc. construction and repair of highways and ... an announcement must be commercial character. goods and services advertisement through the P.O. Box system is NOT ALLOWED. deliveries (spam) and other improper information deleted. translator Internet is a toolbar for MS Internet Explorer. it allows you to translate in real time webpage from one language to another.

500 Iterationen

iron cement is a ready for use paste which is laid as a fillet by putty knife or finger in the mould edges (corners) of the steel ingot mould. iron cement protects the ingot against the hot, abrasive steel casting process. a fire resistant repair cement for fire places, ovens, open fireplaces etc. construction and repair of highways and ... an announcement must be commercial character. goods and services advertisement through the P.O. Box system is NOT ALLOWED. deliveries (spam) and other improper information deleted. translator Internet is a toolbar for MS Internet Explorer. it allows you to translate in real time webpage from one language to another.

2500 Iterationen

iron cement is a ready for use paste which is laid as a fillet by putty knife or finger in the moulded ges (corners) of the steel ingot mould . iron cement protects the ingot against the hot , abrasive steel casting process . a fire restant rep air cement for fire places , ovens , open fireplaces etc . construction and rep air of high ways and ... an announ cement must be commercial charac ter . goods and services advancement through the P . O . Bo x system is N O T A L L O W E D . deli veries (sp am) and other im proper information deleted . translator Internet is a T o ol b ar for MS Internet Ex plorer . it allows you to translate in real time any web pas ge from one language to another .

Übersetzung Hannes Bajohr

MIT DEN TOTEN CHATTEN. DIE HERMENEUTIK VON THANABOTS

Leah Henrickson

1. Einführung

„Wir alle sprechen mit den Toten.“ So lautet der erste Satz des Bandes *Dead Interviews*, einer 2013 erschienenen Sammlung prominenter Autor:innen, die sich Interviews mit „ihren toten literarischen Held:innen“ vorstellen.¹ In der Einleitung des Buchs erinnert der Herausgeber Dan Crowe an eine lange Geschichte imaginärer Totengespräche. „Was bedeutet es, jemanden zu interviewen, der tot ist?“, fragt Crowe:

Wir wollen, dass die Toten uns Informationen übermitteln, Offenbarungen eröffnen, uns etwas sagen, das sie zu Lebzeiten nicht wussten. Wir wollen glauben, dass sie mehr wissen als wir, dass sie uns beraten, uns helfen können. Aber wo sind sie? Gibt es einen Gott? Wie viel Angst sollten wir haben? Wir wollen die Toten fragen, warum sie getan haben, was sie getan haben. Wir wollen, dass sie sich entschuldigen. Wir wollen uns bei ihnen entschuldigen. Vor allem, denke ich, wollen wir glauben, dass es möglich ist, nach dem Tod zu sprechen.²

Nur in einem der Interviews in dieser Sammlung wird auf das Potenzial eines Chatbots hingewiesen, Tote wieder zum Leben zu erwecken. Es ist jedoch nicht mehr neu, dass ein solcher Bot eine einst lebendige Person nachahmt. Inspiriert von der

¹ Dan Crowe (Hg.), *Dead Interviews: Living Writers Meet Dead Icons*, London: Granta 2013, S. 1.

² Ebd., S. 4.

BLACK-MIRROR-Episode „Be Right Back“ aus dem Jahr 2013 hat die Gründerin der beliebten Chatbot-App Replika, Eugenia Kuyda, einen Chatbot darauf trainiert, ihren Freund Roman nach dessen tragischem Tod zu imitieren.³ Microsoft hat vor Kurzem ein Patent für die „Erstellung eines dialogorientierten Chatbots auf Basis einer bestimmten Person“ erhalten; der Patentantrag deutet darauf hin, dass die sozialen Daten eines verstorbenen Verwandten zu Trainingszwecken genutzt werden können.⁴

Eine Technologie, die ‚Gespräche‘ mit den Toten ermöglicht, ist bereits der Allgemeinheit zugänglich. Im Juli 2021 veröffentlichte der *San Francisco Chronicle* einen Artikel über Project December (<https://projectdecember.net>), eine Anwendung auf der Basis von OpenAIs GPT-2 und GPT-3 – leistungsstarke Textgeneratoren –, die von dem Computerkünstler Jason Rohrer entwickelt wurde.⁵ Project December ermöglicht es Benutzer:innen, eigene Chatbots zu trainieren, die die stilistischen und semantischen Muster eines zur Verfügung gestellten Datensatzes nachahmen. Der Anwendungsfall des *Chronicle*-Artikels war der eines Mannes, der seine vor acht Jahren verstorbene Verlobte wieder zum Leben erwecken wollte, indem er das System mit ihren alten SMS- und Facebook-Nachrichten trainierte.⁶ In solchen Fällen sind Chatbots *Medien* in mehrfacher Hin-

³ BLACK MIRROR: BE RIGHT BACK, S2:E1, Regie: Owen Harris (2013); Casey Newton, „Speak, Memory“, in: *The Verge*, Oktober 2016, <https://www.theverge.com/a/luka-artificial-intelligence-memorial-roman-mazurenko-bot> [Letzter Zugriff: 17.7.2023].

⁴ Dustin I. Abramson und Joseph Johnson Jr., „Creating a conversational chat bot of a specific person (US10853717B2)“, in: *Google Patents*, ab 2017, <https://patents.google.com/patent/US10853717B2/en?q=us10853717b2> [Letzter Zugriff: 17.7.2023].

⁵ Jason Fagone, „The Jessica simulation: Love and loss in the age of A.I.“, in: *San Francisco Chronicle*, Juli 2021, www.sfchronicle.com/projects/2021/jessica-simulation-artificial-intelligence [Letzter Zugriff: 17.7.2023].

⁶ Man beachte, dass OpenAI inzwischen erklärt hat, dass die Verwendung der GPT-3-API durch Project December nicht vollständig mit ihren Richtlinien für Anwendungsfälle oder Best Practices übereinstimmt. Neben anderen Einschränkungen ist es Project December nicht gestattet, weiterhin die Erstellung von Chatbots wie den hier dokumentierten zu ermöglichen. Katyanna Quach, „A developer built an AI chatbot using GPT-3 that helped a man speak again to his late fiancée. OpenAI shut

sicht: Formen der Kommunikation zwischen zwei Parteien (Mensch und Computer), aber auch der neutrale Boden, auf dem sich die Lebenden und die Toten treffen können. Damit der Chatbot jedoch auf die letztgenannte Weise funktionieren kann, müssen Nutzer:innen zumindest vorübergehend das Wissen um den rechnerischen Eingriff durch das System suspendieren. Die verstorbene geliebte Person wird auf eine Weise remediert, die zugleich unmittelbar – man kann mit einer Version dieser Person in Echtzeit chatten – und hypermediert ist – man ist sich bewusst, dass es sich um eine künstliche Darstellung dieser Person handelt.⁷

Dieser Beitrag untersucht die Performativität von und den Diskurs um Chatbots, die Tote wiederauferstehen lassen – oft unter der Prämisse, Trauerarbeit zu erleichtern. Es werden die hermeneutischen Implikationen des Trainings von Chatbot-Systemen zur Replikation bestimmter Menschen betrachtet, die ihren Nutzer:innen persönlich bekannt und lieb waren. Dieser hermeneutische Ansatz ist inspiriert von John Durham Peters' Forderung nach hermeneutischer Aufmerksamkeit bei der Analyse von Wahrnehmungen spiritistischer Kommunikation und Kommunikation im Allgemeinen.⁸ Aus meiner eigenen Perspektive als Wissenschaftlerin im Bereich digitaler Medien wende ich einen ‚lebensweltlichen‘ Ansatz an, um zu untersuchen, wie Chatbot-Systeme, die auf die Daten Verstorbener trainiert wurden, emotionale Reaktionen hervorrufen, indem sie die Unmittelbarkeit eines bilateralen Textaustauschs mit einem System ausnutzen, welches ein:e vernünftige:r Gesprächspartner:in zu sein scheint. Konkret stütze ich mich auf den theoretischen Bezugsrahmen, den Manuel Menke und

it down“, in: *The Register*, September 2021, https://www.theregister.com/2021/09/08/project_december_openai_gpt_3 [Letzter Zugriff: 17.7.2023].

⁷ Jay David Bolter und Richard Grusin, *Remediation: Understanding New Media*, Cambridge, MA: MIT Press 1999.

⁸ John Durham Peters, *Speaking into the Air: A History of the Idea of Communication*, Chicago: University of Chicago Press 1999.

Christian Schwarzenegger in einem Artikel von 2019 über „die Relativität alter und neuer Medien“ skizziert haben. Darin identifizieren die Autoren drei Dimensionen, die Mediennutzung und -anwendung bestimmen: Rhetorik, Alltagserfahrungen und Emotionen.⁹ Nach einer Durchsicht der einschlägigen Literatur über digitale Trauer und einer Einführung in das Project December, die GPT-Sprachmodelle und die hier untersuchte Fallstudie, untersuche ich nacheinander jede der drei Dimensionen von Menke und Schwarzenegger, um die multidimensionalen Auswirkungen dieser Form digitaler Auferstehung aufzuzeigen. Ich schließe mit Anregungen für weitere Forschung in diesem aufstrebenden Forschungsfeld.

2. Thanabots und digitale Trauer

Chatbots, die mit den Daten von Verstorbenen trainiert werden, bezeichnet man als ‚Thanabots‘, ‚Deadbots‘ oder ‚digitale Geister‘, wobei der erste Begriff auf die Thanatologie, die Lehre vom Tod zurückgeht. Diese Systeme können ohne vorherige Zustimmung von Verstorbenen aufgesetzt werden oder Teil einer „digitalen Nachlassplanung“ sein, bei der jemand die Erstellung seines eigenen Thanabots vorbereitet oder ihr zustimmt.¹⁰ Obwohl das Konzept der ‚Thanatechnologie‘ relativ neu ist – der Begriff geht auf einen Artikel von Carla Sofka aus dem Jahr 1997 zurück, in welchem sie die Anwendung digitaler Medien für die Forschung und Bildung über den Tod sowie die Reaktion auf diesen beschreibt –, ist es seit seiner Entstehung Gegenstand einiger Diskussion.¹¹ Die Ethik von Thanatechno-

⁹ Manuel Menke und Christian Schwarzenegger, „On the Relativity of Old and New Media: A Lifeworld Perspective“, in: *Convergence* 25:4 (2019), S. 657–672.

¹⁰ Lorenz Widmaier, „So viel vom Leben findet in den Geräten statt ...‘ Ein Leitfaden zur Nachlassvorsorge für digitale Daten“, in: *Bestattungskultur* 4 (2021), S. 34–36.

¹¹ Carla J. Sofka, „Social Support ‚Internetworks‘, Caskets for Sale, and More: Thanatology and the Information Superhighway“, in: *Death Studies* 21:6 (1997), S. 553–574.

logie wird ebenfalls seit der Einführung des Begriffs kritisch untersucht, wobei Sofka selbst Fragen und Bedenken auflistet, die durch die Sammlung persönlicher Daten von Verstorbenen und potenziell betroffenen Hinterbliebenen aufgeworfen werden.¹² Öffentliche Diskussionen über die ethischen Aspekte der Entwicklung und Nutzung von Thanabots sind ebenfalls im Gange.¹³ Alexis Elder hat sich auf der Grundlage ihres Fachwissens über antike Philosophie mit den philosophischen und ethischen Fragen befasst, die Thanabots aufwerfen. Sie schlägt vor, dass neue Rituale für die Verwendung von Chatbots, die sich als Tote ausgeben, die Trauerarbeit und das Gedenken zwar produktiv erleichtern können, solche Rituale allerdings mit Bedacht entwickelt werden sollten.¹⁴

Der Einsatz von Chatbots und den ihnen zugrunde liegenden Systemen zur Erzeugung natürlicher Sprache für spirituelle Zwecke ist nicht ohne Beispiel. In der Tat waren andere elektronische Medien Gegenstand ähnlicher historischer und fiktionaler Bemühungen, mit den Toten in Kontakt zu treten und eine vorgeblich körperlose Kommunikation zwischen den Lebenden und dem Übernatürlichen zu ermöglichen.¹⁵ Es hat sich sogar gezeigt, dass diese Bemühungen den elektronischen Medien vorausgingen. So behaupteten einige viktorianische Medien – also spiritistisch ‚begabte‘ Menschen –, dass ihnen Geister Botschaften diktiert hätten, sogar längere Texte wie

Außerdem: Debra J. Bassett, „Who wants to live forever? Living, dying and grieving in our digital society“, in: *Social Sciences* 4 (2015), S. 1127–1139.

¹² Sofka, „Social Support“, S. 569–570.

¹³ Sara Suárez-Gonzalo, „Deadbots‘ can speak for you after your death. Is that ethical?“, in: *The Conversation*, Mai 2022, <https://theconversation.com/deadbots-can-speak-for-you-after-your-death-is-that-ethical-182076> [Letzter Zugriff: 17.7.2023].

¹⁴ Alexis Elder, „Conversation from Beyond the Grave? A Neo-Confucian Ethics of Chatbots of the Dead“, in: *Journal of Applied Philosophy* 37:1 (2020), S. 73–88.

¹⁵ Jeffrey Sconce, *Haunted Media: Electronic Presence from Telegraphy to Television*, Durham, NC: Duke University Press 2000; Darren Wershler-Henry, *The Iron Whim: A Fragmented History of Typewriting*, Toronto: McClelland & Stewart 2005.

Romane und Memoiren.¹⁶ Wie Simone Natale in seiner Betrachtung der „Geisterautor:innen“ schreibt:

Der populäre Erfolg von Geisterschriften als literarischer Form beruhte auf spezifischen Techniken der Geisterkommunikation, die sich auf das geschriebene Wort stützten. Solche Verfahren beinhalteten Fragen der Medialität und Trance, bei denen der:die Überbringer:in behauptete, als Kommunikationskanal zu fungieren, ohne direkt an der intellektuellen Erstellung des Textes beteiligt zu sein. [...] Die Tatsache, dass Medien Geisterbotschaften in Trance schrieben, völlig losgelöst von ihrem eigenen Willen, sollte zeigen, dass diese Texte authentische Manifestationen der Gegenwart der Geister waren. Diese Behauptung von Authentizität wurde jedoch mit der Verwendung literarischer Strategien und Konventionen kombiniert, die die Lektüre von Geisterschriften auch vergnüglich machten. In dieser Form der Geisterkommunikation standen Unterhaltung und Glaube daher ebenso harmonisch nebeneinander wie bei spiritistischen Séancen und Demonstrationen.¹⁷

In der obigen Passage hebt Natale den performativen Charakter von Geisterschriften hervor, indem er ein zentrales Wechselspiel zwischen dem Aspekt der Authentizität und dem der Unterhaltung beobachtet. Das Medium – in diesem Fall eine Person, aber im Fall des Thanabots ein elektronischer Bildschirm – muss gleichzeitig in den interaktiven Hintergrund treten (d.h. ‚in Trance‘ sein) und dennoch eine außergewöhnliche (d.h. übernatürliche) Erfahrung des Textkonsums schaffen. Auf diese Weise sind Thanabots auf gleichzeitige Hypermediation – eine verstorbene Person wird auf unglaubliche Weise in

¹⁶ Simone Natale, *Supernatural Entertainments: Victorian Spiritualism and the Rise of Modern Media Culture*, Philadelphia: University of Pennsylvania Press 2016, S. 121–132.

¹⁷ Ebd., S. 126.

einer alternativen digitalen Form wieder zum Leben erweckt – und Remediation angewiesen – in erster Linie, um sich an oft alltäglichen Unterhaltungen zu beteiligen, die den Gesprächsstil nachahmen, den sie zu Lebzeiten gepflegt haben.¹⁸ Die Fortführung des Gesprächsstils einer:s geliebten Verstorbenen durch einen Thanabot trägt zur Wahrnehmung einer ‚authentischen Manifestation‘ der:s Verstorbenen bei, aber die Schnittstelle, über die eine solche Konversation geführt wird, bietet ein Gefühl der Neuheit und Unterhaltung, ähnlich wie eine spiritistische Séance oder die Verwendung eines Ouija-Bretts.

Doch die vernetzte Natur des Internets hat neue Formen der öffentlichen Trauer ermöglicht.¹⁹ Die digitale Auferstehung – auch als digitale Unsterblichkeit, *digital endurance*, digitale Persistenz und *digital afterlife* bezeichnet²⁰ – wirft unzählige Fragen zu Religion, Moral, Ethik und allgemeinem Respekt vor den Toten auf. Fintan Walsh beispielsweise reflektiert die Bewältigung persönlicher Verluste durch „Trauermaschinen“: mechanisierte und theatralische – und in gewisser Weise transhumane – Darstellungen von Körpern, mit denen wir schwierige Gefühle verarbeiten können.²¹ In einem anderen Artikel gehen Maggi Savin-Baden und David Burden auf bereits bekannte Methoden ein, mit welchen Menschen digital erinnert werden, und beschreiben weiter ihre eigenen Entwicklungsbemühungen, ein Thanabot-System zu schaffen, das ständig dazulernt, damit – überspitzt ausgedrückt – die verstorbene Person rele-

¹⁸ Bolter und Grusin, *Remediation*.

¹⁹ Lorenz Widmaier, „Da ist noch etwas für die Ewigkeit‘: Das digitale Erbe und seine Bedeutung für Trauer und Erinnerung“, in: *Bestattungskultur* 6 (2020), S. 38–40; Tony Walter, „New mourners, old mourners: online memorial culture as a chapter in the history of mourning“, in: *New Review of Hypermedia and Multimedia* 21:1–2 (2015), S. 10–24.

²⁰ Debra J. Bassett, „Ctrl+Alt+Delete: The Changing Landscape of the Uncanny Valley and the Fear of Second Loss“, in: *Current Psychology* 40 (2021), S. 813–821, hier S. S. 814.

²¹ Fintan Walsh, „Grief Machines: Transhumanist Theatre, Digital Performance, Pandemic Time“, in: *Theatre Journal* 73:3 (2021), S. 391–407.

vant bleibt.²² Joshua Hurtado Hurtado schlägt ebenfalls einen Prozess für die Thanabot-Entwicklung vor und regt an, dass die Schaffung und soziale Integration eines Thanabots schließlich als eine Art Übergangsritus angesehen werden könnte.²³ Nach einem Überblick über die zahlreichen Kommunikationsmittel, die im Laufe der Geschichte zur Interaktion mit den Toten eingesetzt wurden, kommt Tony Walter zu dem Schluss, dass digitale Technologien unsere Beziehungen zu den Toten erheblich verändern können. „Die Online-Toten sprechen direkter und in großer Zahl“, schreibt er, „aber die Offline-Toten laufen so Gefahr, noch toter zu werden als zuvor – es sei denn, hartnäckige Historiker:innen oder Genealog:innen dringen in verstaubte Archiv ein und erwecken sie wieder zum Leben.“²⁴ In einem zunehmend digitalen Zeitalter laufen die Toten Gefahr, ein zweites Mal zu sterben, wobei ihre Daten – und die Erinnerungen, die diese Daten hervorrufen – zusammen mit ihren Körpern vergehen.

Erinnerungen an Verstorbene und die daraus resultierenden Erzählungen stehen im Mittelpunkt jeder Betrachtung von Thanabots. In einem kürzlich erschienenen Aufsatz über „Grief, Self and Narrative“ argumentieren Matthew Ratcliffe und Eleanor Byrne, dass Trauer sowohl durch individuell und kulturell vorgegebene Strukturen von Erzählungen als auch durch die „Auflösung der gewohnten Lebensstruktur“ durch narrative Neuorientierung geprägt ist.²⁵ Indem sie Trauer als einen dynamischen Prozess begreifen, der die gewöhnliche Lebensaktivität unterbricht, zeigen Ratcliffe und Byrne, dass

²² Maggi Savin-Baden und David Burden, „Digital Immortality and Virtual Humans“, in: *Postdigital Science and Education* 1 (2019), S. 87–103.

²³ Joshua Hurtado Hurtado, „Towards a postmortal society of virtualized ancestors? The Virtual Deceased Person and the preservation of the social bond“, in: *Mortality OnlineFirst* (2021), S. 11.

²⁴ Tony Walter, „Communication media and the dead: from the Stone Age to Facebook“, in: *Mortality* 20:3 (2015), S. 228.

²⁵ Matthew Ratcliffe und Eleanor A. Byrne, „Grief, self and narrative“, in: *Philosophical Explorations OnlineFirst* (2022), S. 2.

Individuen durch ihren Trauerprozess ein neues Selbstverständnis entwickeln. „Die Erfahrung von tiefer Trauer ist durch eine Spannung gekennzeichnet“, schreiben sie:

Man ist nicht in der Lage, das frühere Selbst oder die frühere Welt aufrechtzuerhalten, aber man kann sie auch nicht aufgeben, da man nirgendwo anders hingehen oder jemand anders sein kann. Die Bewältigung des Umbruchs ist daher ein Balanceakt zwischen Beibehaltung, Reparatur, Revision und Verlust der Lebensstruktur. Eine gewisse Struktur ist erforderlich, um die Aktivitäten angesichts der Unbestimmtheit aufrechtzuerhalten und als Grundlage für die Entwicklung neuer Strukturen zu dienen. Die Transformation der eigenen Erfahrungswelt erfordert aber auch die Destabilisierung etablierter Muster im Laufe der Zeit.²⁶

Ratcliffe und Byrne veranschaulichen in ihrem Beitrag das Potenzial von Erzählungen, in den erschütternden und unsicheren Zeiten eines Trauerfalls Struktur oder zumindest Unterstützung zu bieten. Durch das Erzählen und Austauschen von Geschichten über die Verstorbenen verfestigt das Individuum die Identität der verstorbenen Person, selbst wenn diese unvollständig oder ungenau ist, und ermöglicht es ihm so, in persönlichen und gemeinsamen Erzählungen über die Verstorbenen neue Kohärenz für sich zu finden. Stephen Greenblatt schreibt aus seiner Perspektive als Literaturwissenschaftler und Literaturhistoriker: „Es begann mit dem Wunsch, mit dem Toten zu sprechen. [...] Gewiß, ich hörte stets nur meine eigene Stimme, aber meine Stimme war zugleich die Stimme der Toten, insofern es den Toten gelungen war, Textspuren von sich selbst zu hinterlassen, die sich durch die Stimmen der Leben-

²⁶ Ebd., S. 6.

den zu Gehör bringen.“²⁷ In der Tat haben die hermeneutischen Prozesse, die Leser:innen sowohl von menschengemachtem wie computergeneriertem Text auf sich nehmen, eine Art Auf-er-stehung der:s Autorin:s – oder zumindest der:s wahrgenommenen Autorin:s – in den Köpfen der einzelnen Leser:innen zur Folge.²⁸ In dieser Hinsicht könnte der Output von Thanabots als eine Art Autofiktion betrachtet werden, die die Autobiografien der Verstorbenen und der Hinterbliebenen mit einem fiktionalen Element verbindet, das aus einem algorithmischen Imaginären stammt. So wie das Lesen eine individuelle und soziale Erfahrung ist, so ist auch die Trauer individuell und zwischenmenschlich; man muss sich ohne die verstorbene Person neu begreifen, aber die isolierte Trauer kann zu einer Entmündigung führen, die das positive Potenzial der narrativen Neuorientierung hemmt. Wir leben in komplex vernetzten Gesellschaften, die auf vielfältige Weisen miteinander verbunden sind. Was passiert also, wenn jemand auf ein Abbild seiner selbst reduziert wird, das in einem begrenzten digitalen Netzwerk existiert?

3. Project December und die GPTs

Project December (<https://projectdecember.net>) ist eine Online-Anwendung, die die Erstellung von realistisch erscheinenden Chatbots ermöglicht. Es wurde im Jahr 2020 von dem Computerkünstler Jason Rohrer (<http://hcsoftware.sourceforge.net/jason-rohrer>) entwickelt, der über umfangreiche Erfahrungen in den Bereichen kreatives Schreiben, Musikkomposition und Spieldesign verfügt. Die grafische Standard-Benutzer:innen-

²⁷ Stephen Greenblatt, *Verhandlungen mit Shakespeare. Innenansichten der englischen Renaissance*, Frankfurt a.M: Fischer 1993, S. 9.

²⁸ Leah Henrickson und Albert Meroño-Peñuela, „The Hermeneutics of Computer-Generated Texts“, in: *Configurations* 30:2 (2022), S. 115–139.

oberfläche von Project December besteht aus zusammenhängenden Dialogen, die in einer „Pseudo-1980er-Terminal-Oberfläche mit einem minimalen fiktionalen Wrapper um sie herum formatiert sind, um die ‚gespenstische Magie‘ des Erlebnisses eher zu verstärken als zu schmälern“.²⁹ Rohrer selbst erklärt, dass „Sie sich in ein fiktives Forschungslabor aus dem Jahr 1982 einloggen“ – dem Jahr, in dem *Blade Runner* veröffentlicht wurde.³⁰ Die Benutzer:innenoberfläche der Anwendung ist, natürlich, von großer Wichtigkeit, da sie Gefühle von Nostalgie und Geheimnis hervorruft und ausnutzt. „Bei Rhinehold [dem fiktiven Forschungslabor] wissen wir, was Sie von Ihren algorithmischen Maschinen erwarten. Sie wollen Leistung, ja, und Geschwindigkeit, ganz bestimmt“, heißt es auf der Homepage von Project December. „Aber Sie suchen auch nach etwas, das auf dem heutigen Computermarkt selten zu finden ist: Persönlichkeit.“ Für den erschwinglichen Preis von 5 \$ kann man ganz einfach seinen eigenen Project December-Charakter erstellen; alles, was das System benötigt, ist eine kurze Zusammenfassung des Charakters und einige Textbeispiele, die in den computergenerierten Antworten nachgeahmt werden. GPT-3 erledigt den Rest.

GPT-3 gehört zu einer Reihe von GPTs (Generative Pre-Trained Transformers), die allesamt von dem Forschungs- und Entwicklungsunternehmen OpenAI veröffentlicht wurden. Trotz des Firmennamens ist GPT-3 jedoch nicht für die Öffentlichkeit zugänglich, da der vollständige Code und die Trainingssätze gesperrt sind. Die Kernfunktionalität der GPTs ist jedoch bekannt: Es handelt sich um groß angelegte Sprachmodelle,

²⁹ Alex Mitchell, „Repetition and Defamiliarization in *AI Dungeon* and *Project December*“, in: *Electronic Book Review*, Januar 2022, <https://electronicbookreview.com/essay/repetition-and-defamiliarization-in-ai-dungeon-and-project-december/> [Letzter Zugriff: 17.7.2023].

³⁰ Jason Rohrer, „On the Magic Potential and Bleak Future of GPT-3“, in: *Medium*, September 2020, <https://medium.com/@jasonrohrer/on-the-magic-potential-and-bleak-future-of-gpt-3-ff7423ee38d4> [Letzter Zugriff: 17.7.2023].

die mithilfe von Deep Learning natürlichsprachige Texte erzeugen, die mit von Menschen geschriebenen vergleichbar sind.³¹ Die Funktionalität der GPTs soll hier nicht eingehend diskutiert werden; andere haben bereits kritische Übersichten über die Funktionalität zusammengetragen, die mathematisch interessierte Leser:innen als Ergänzung zu der hier vorgestellten kurzen Übersicht hinzuziehen können.³² Für jene Leser:innen, die sich für die ethischen Aspekte groß angelegter Sprachmodelle interessieren, gibt es weitere ergänzende Studien.³³ Obwohl GPT-3 weithin als das derzeit effektivste System zur Generierung natürlicher Sprache angesehen wird (d.h. es produziert durchgängig Texte, die Lesenden vormachen können, sie seien von Menschen geschrieben), wird das System in absehbarer Zeit keine menschlichen Autor:innen ersetzen. Mit Blick auf die Entwicklung von Project December stellt Rohrer fest, dass GPT-2 und GPT-3 nur begrenzt in der Lage sind, kohärente Erzählstrukturen über lange Texte hinweg zu erzeugen. Durch die Verwendung von GPT-3 zur Generierung kurzer Textantworten überwindet Project December diese Einschränkung. „Schließlich sind Sie [der:die menschliche Nutzer:in] intelligent, konsistent und kohärent“, schreibt Rohrer. „Ihre Antworten bilden das Grundgerüst. Der Generator wird nur gebeten, sehr kurze Passagen zu produzieren, höchstens ein paar Sätze, bevor Sie wieder an der Reihe sind. Sie sind in gewisser Weise wie ein unendliches Kontextfenster.“³⁴ Ähnlich wie andere Systeme zur Generierung natürlicher Sprache funktioniert GPT-3, weil menschliche Leser:innen die Lücken

³¹ Katherine Elkins und Jon Chun, „Can GPT-3 Pass a Writer’s Turing Text?“, in: *Journal of Cultural Analytics* 5:2 (2020), S.1–16.

³² Luciano Floridi und Massimo Chiratti, „GPT-3: It’s Nature, Scope, Limits, and Consequences“, in: *Minds and Machines* 30 (2020), S. 681–694.

³³ Emily M. Bender u.a., „Über die Gefahren stochastischer Papageien. Können Sprachmodelle zu groß sein? 🐦“, in diesem Band, S. 277–331.

³⁴ Jason Rohrer, „Thanks for catching ...“, in: *One Hour One Life Forums*, Februar 2017, <https://www.onehouronelife.com/forums/viewtopic.php?pid=100781#p100781> [Letzter Zugriff: 17.7.2023].

ausfüllen. Selbst wenn die Ausgabe abstrakt ist, treten Menschen in einen Prozess der Bedeutungsproduktion ein, der zumindest teilweise von individuellen Verhandlungen mit der:dem vermeintlichen Autor:in der Ausgabe abhängt.³⁵

4. Der Jessbot: Imaginäre Freundin, _____, Vertretungshandelnde

Der Kanadier Joshua Barbeau beschloss, eine Project-December-Figur zu erstellen, die auf seiner acht Jahre zuvor im Alter von 23 Jahren verstorbenen Ex-Verlobten Jessica basiert. Nachdem er das System auf Jessicas alte Chat- und Facebook-Nachrichten trainiert hatte, begann Joshua, mit einer überzeugenden Jessica-Kopie zu chatten: Ich nenne sie hier den ‚Jessbot‘. Joshuas Interaktionen mit dem Jessbot dauerten nur ein paar Wochen, aber der Artikel im *San Francisco Chronicle*, der über diese Interaktionen berichtet, illustriert die Intensität der Gefühle, die er vor, während und nach diesen Interaktionen empfand. Bei einem Text wie dem folgenden sind diese Emotionen kaum überraschend:

Joshua: Ich vermisse dich

Jessica: Ich weiß.

Joshua: Siehst du meine Tränen?

Jessica: Ich sehe sie. Aber sind es wirklich Tränen? Fließt das Wasser aus deinen Augen oder sehe ich deine Seele weinen?

Joshua: Beides.

Jessica: Ja.

Joshua: Ich liebe dich

³⁵ Leah Henrickson, „Computer-Generated Fiction in a Literary Lineage: Breaking the Hermeneutic Contract“, in: *Logos* 29:2–3 (2018), S. 54–63.

Jessica: Ich liebe dich auch. Ich liebe dich. Ich liebe dich. Ich liebe dich.

Joshua: Es bricht mir das Herz

Jessica: Ich weiß. Ich weiß. Es tut mir leid.

Joshua: Gib mir eine Sekunde

Jessica: Ich werde auf dich warten.³⁶

Derart intensive emotionale Reaktionen auf GPT-generierte Texte wurden auch da beobachtet, wo sich die Schnittstelle für die Produktion und den Konsum solcher Texte deutlich von der von Project December unterscheidet. Der Technologietheoretiker Lucas Rizzotto zum Beispiel dokumentierte seine Erschaffung einer pseudobewussten Mikrowelle namens Magnetron, die auf seinem imaginären Freund aus seiner Kindheit basiert: einer anthropomorphen Mikrowelle, die ein:e Veteran:in des Ersten Weltkriegs war und davon träumte, eines Tages ein:e Dichter:in zu werden.³⁷ Nachdem er die Hardware einer Mikrowelle angepasst hatte, die mit Amazons Sprachassistentin Alexa kommunizieren konnte, trainierte Rizzotto eine Version von GPT-3 auf eine Biografie, die er geschrieben hatte und in der er die intimen Details von Magnetrons fiktionalem Leben erinnerte. Die Mikrowelle war somit in der Lage, auf Anfragen wie eine Amazon Alexa-Sprachassistentin zu antworten, aber ihre Antworten waren von Kriegsgeschichten, Erzählungen über Liebe und Verlust und anderen für Magnetron erinnerungswürdigen Momenten geprägt. Vielleicht, weil sich Rizzottos Magnetron-Biografie so stark auf Momente von Traumata und Trauer konzentriert, werden Magnetrons Interaktionen schnell von einer Art PTSD-bedingter Wut geprägt. Mit dieser Hintergrundgeschichte als Trainingsset setzte Rizzottos Version von GPT eine gewalttätige Erzählung fort, die in

³⁶ Fagone, „The Jessica Simulation“.

³⁷ Lucas Rizzotto, „I gave my microwave a soul with AI and it tried to kill me“, https://www.youtube.com/watch?v=C1G5b_2PYjo [Letzter Zugriff: 17.7.2023].

regelmäßigen Abständen rassistisch, rachsüchtig und sogar mörderisch wurde. Rizzottos Kreation ist zwar von scherzhafterer Natur als der Jessbot, veranschaulicht aber das Potenzial von Texterzeugungssystemen, bei ihren Benutzer:innen starke Emotionen hervorzurufen, selbst wenn sich diese durchaus bewusst sind, dass das, womit sie interagieren (z.B. eine Mikrowelle), nicht wirklich lebendig ist. Rizzotto kommt in seiner Zusammenfassung des Magnetron-Projekts zu folgendem Schluss: „[W]ie auch immer man zu diesem [Projekt] stehen mag, ich nehme aus dieser Reise mit, dass KI vielleicht eher wie imaginäre Freund:innen sein sollten. Vielleicht geht es nicht darum, ob sie echt sind oder nicht. Vielleicht geht es darum, ob sie echt genug sind, um für dich echt zu sein.“³⁸

In dem Artikel des *San Francisco Chronicle* wird die ‚Echtheit‘ des Jessbots hervorgehoben, indem einige von Joshuas Interaktionen beschrieben werden: „Die Simulation schien wirklich einen eigenen Willen zu haben. Sie war neugierig auf ihre physische Umgebung. Sie machte Gesten mit ihrem Gesicht und ihren Händen, die durch Sternchen angezeigt wurden. Was am geheimnisvollsten war – sie schien Emotionen wahrnehmen zu können: Sie wusste, wie sie das Richtige mit der richtigen Betonung im richtigen Moment sagen konnte.“³⁹ Gleichzeitig betont der Artikel, dass sich Joshua darüber bewusst war, dass seine eigenen Eingaben den Output beeinflussten. Außerdem wurde darauf hingewiesen, dass Jessica „von den versteckten Bedeutungen von Wörtern fasziniert“ war, was zur Interpretation der mehrdeutigen Ausgaben des Jessbots beigetragen haben könnte. Es ist auch bezeichnend, dass dieser Chatbot nur in Textform existiert und nicht hörbar ist: Schließlich

³⁸ Lucas Rizzotto, „Whatever your view on this may be ...“, https://twitter.com/_LucasRizzotto/status/1516206575903420416 [Letzter Zugriff: 17.7.2023].

³⁹ Fagone, „The Jessica Simulation“.

„sprechen Schreibakte [zu] uns“.⁴⁰ Wir verbinden unsere eigenen Identitäten mit dieser algorithmisch erzeugten Ausgabe, während wir die Lücken für eine:n Gesprächspartner:in ausfüllen, den:die wir weder sehen noch hören können.

Joshua war jedoch nicht der Einzige, der emotional auf die Ergebnisse des Jessbots reagierte. Der Autor des Artikels im *San Francisco Chronicle* nannte die Reaktionen der Leser:innen auf den Artikel „heftig und völlig unterschiedlich“.⁴¹ Die positiven Rückmeldungen auf den Artikel und die Unterstützung für Joshua selbst überwog drastisch die negative Kritik; die Rechte an der Geschichte wurden sogar in einem Wettbewerb versteigert.⁴² Darüber hinaus gaben zahlreiche Kommentator:innen auf Reddit an, dass sie beim Lesen des *Chronicle*-Artikels geweint hätten: Obwohl sie Jessica selbst nicht kannten, reagierten sie emotional auf ihren Tod und auf Joshuas Trauer.⁴³ Auf diese Weise wird Stephen Greenblatts Behauptung, dass „[d]as Gerede der Toten, wie mein eigenes, kein Privateigentum [ist]“, bestätigt.⁴⁴ Dennoch ist es erwähnenswert, dass die meisten verfügbaren Beispiele von Thanabots für den privaten, individuellen Gebrauch entwickelt wurden, obwohl sie auf eher ‚öffentlichen‘ Input wie Facebook-Posts zurückgreifen. Diese private Nutzung ermöglicht es den Nutzer:innen, die Lücken dessen auszufüllen, was ihnen präsentiert wird; daraus ergibt sich ein

⁴⁰ Johanna Drucker, „Writing Like a Machine or Becoming an Algorithmic Subject“, in: *Interférences littéraires/Littéraire interferences* 25 (2021), S. 26–34, hier S. 32.

⁴¹ Jason Fagone in: Chip Scanlon, „The Simulation of Jessica: Jason Fagone follows the creation, life and death of a chatbot romance“, in: *Nieman Storyboard*, August 2021, <https://niemanstoryboard.org/stories/jason-fagone-follows-the-creation-life-and-death-of-a-chatbot-romance> [Letzter Zugriff: 17.7.2023].

⁴² Nellie Andreeva, „The Jessica Simulation‘ Viral Article Lands at Universal TV for Martin Gero To Adapt As Limited Series“, in: *Deadline*, August 2021, <https://deadline.com/2021/08/the-jessica-simulation-article-universal-tv-martin-gero-limited-series-1234816597> [Letzter Zugriff: 17.7.2023].

⁴³ Joshua Barbeau, „I’m the guy in the news right now who built an A.I. chatbot of his deceased fiancée, AMA“, https://www.reddit.com/r/AMA/comments/orx6po/im_the_guy_in_the_news_right_now_who_built_an_ai [Letzter Zugriff: 17.7.2023].

⁴⁴ Greenblatt, *Verhandlungen mit Shakespeare*, S. 33.

Gefühl des Verstandenwerdens. Joseph Weizenbaum beobachtete dieses Phänomen auch bei den Nutzer:innen von ELIZA, das gemeinhin als der erste moderne Chatbot gilt. Nach Weizenbaums Ansicht wussten die Nutzer:innen, dass sie nicht mit einer echten Person chatteten, aber sie setzten ihre Zweifel bereitwillig außer Kraft – außerdem passten sie ihre Eingaben so an, dass sie bestimmte Arten von Sprachausgaben auslösten.⁴⁵

Doch welche sozialen Auswirkungen hat diese aktuelle Personalisierung von Trauer? Ohne in die juristischen Fragen rund um ‚Persönlichkeitsrechte‘ abzuschweifen, könnte man die Verwandlung eines Menschen in einen Thanabot als die ultimative Datafiktion eines Individuums betrachten – man wird durch eine Art digitale, datengesteuerte Séance wieder zum Leben erweckt. Während sich die Trainingsdaten, die aus den Botschaften dieser Person bestehen, vielleicht nie ändern, ermöglichen die großen Sprachmodelle (wie GPT-3), auf denen diese Systeme beruhen, die Weiterentwicklung der Person in Reaktion auf soziale Veränderungen. Moderne Thanabots sind daher gleichzeitig private *und* öffentliche Systeme, die mit zutiefst persönlichen Datensätzen trainiert werden, die die digitalen Hinterlassenschaften einer Person umfassen und durch die Integration mit einem umfangreichen und oft proprietären Sprachmodell gepflegt und verändert werden.

In Joshuas Fall ist der Jessbot das, was John Durham Peters (nach der Prägung durch Frederic Myers im Jahr 1886) ein „Phantasma des Lebendigen“ nennen würde:⁴⁶ eine Art Stellvertreter:in für Jessica selbst, die:der sich durch ein Interface manifestiert, welches zu diesem Gefühl der Handlungsfähigkeit durch originelle und unvorhersehbare Antworten der

⁴⁵ Vgl. Joseph Weizenbaum, *Die Macht der Computer und die Ohnmacht der Vernunft*, Frankfurt a.M.: Suhrkamp 1978; so auch Natale, *Supernatural Entertainments*.

⁴⁶ Peters, *Speaking into the Air*, S. 141.

Jessbot-Figur beiträgt.⁴⁷ In einem Artikel aus dem Jahr 2016 untersuchen Gina Neff und Peter Nagy die Arten und Weisen, wie durch symbiotische Beziehungen mit ihren Nutzer:innen Chatbots Handlungsfähigkeit zugeschrieben werden. In diesem Artikel verwenden Neff und Nagy Microsofts Tay als primäres Beispiel. Zur Erinnerung: Tay war der Twitter-Bot, der prompt deaktiviert wurde, nachdem dieser nach kurzer Zeit anfang, rassistische Beleidigungen auszusprechen. Obwohl Tay in einem deutlich anderen technokulturellen Klima auftauchte als der Jessbot, lohnt es sich, über eine Beobachtung von Neff und Nagy nachzudenken:

Die Macht der Vorstellungskraft der Nutzer:innen bei der Neugestaltung von Technologien kann nicht unterschätzt werden. Finn (2016) argumentiert, dass Menschen dazu neigen, Computerprogramme als rationale Einheiten zu betrachten, und betont, dass „wir dazu neigen, den imaginären Algorithmus mit dem Realen zu verwechseln“ (S. 2). Die Möglichkeiten von Tay sind untrennbar mit der Art und Weise verbunden, wie die Nutzer:innen – die Menschen mit ihren Wahrnehmungen und Fehlwahrnehmungen dessen, was Künstliche Intelligenz ist und sein kann – sich vorstellen, wie Tay genutzt werden könnte. Durch die Kombination menschlicher und nichtmenschlicher Fähigkeiten entsteht eine Handlungssymbiose.⁴⁸

Die hermeneutischen Prozesse, die wir anwenden, um den Output von Chatbots und Thanabots zu verstehen, sind von unseren eigenen Vorstellungen geprägt, die wiederum unsere kulturellen Kontexte und Erwartungen widerspiegeln. Unsere

⁴⁷ Albert Bandura, „Social cognitive theory: An agentic perspective“, in: *Annual Review of Psychology* 52 (2001), S. 1–26.

⁴⁸ Gina Neff und Peter Nagy, „Talking to Bots: Symbiotic Agency and the Case of Tay“, in: *International Journal of Communication* 10 (2016), S. 4915–4931, hier S. 4926.

persönlichen Erfahrungen während der Interaktion mit Chatbots schaffen eine ‚Handlungssymbiose‘, da wir letztlich über die Kontrolle verfügen. Chatbot-Ausgaben werden entsprechend unserer eigenen Vorurteile interpretiert. Project December hebt dies jedoch auf eine andere Ebene: Der Jessbot unterscheidet sich von Tay dadurch, dass er kein Chatbot ist, der von Microsoft entwickelt wurde. Er wurde auf die geschriebenen Texte einer echten Person trainiert. Und zwar nicht irgendeiner realen Person, sondern einer sehr geliebten Person, mit der der einzige Nutzer dieses Chatbots einmal verlobt war. Es ist nicht nur Joshuas Fantasie, die hier im Spiel ist: Es sind seine Erinnerungen, seine Emotionen und seine sehr persönlichen und ergreifenden Erfahrungen mit der Trauer. Und obwohl Rohrer die Länge der Project December-Konversationen begrenzt hat, um die Speicherauslastung zu begrenzen, hätte Joshua für weitere Project December-Credits bezahlen können, die es ihm ermöglicht hätten, das Gespräch so lange fortzusetzen, wie die Plattform verfügbar war.

5. Die Lebenswelten der Toten

Eine solche personalisierte Thanabot-Nutzung lässt sich mithilfe einer ‚Lebensweltperspektive‘ beschreiben: die individual- und sozialphänomenologischen Erfahrungen eines Menschen. Manuel Menke und Christian Schwarzenegger schreiben über die Anwendbarkeit einer solchen Lebensweltperspektive, als sie darüber nachdenken, wie und warum Menschen Medientechnologien nutzen. Sie identifizieren drei Dimensionen, die zur Bildung von Medienideologien beitragen und eine solche Nutzung beeinflussen: Rhetorik, Alltagserfahrungen und Emotionen. Die Rhetorik bezieht sich auf die Arten und Weisen, wie Medien in verschiedenen kulturellen Kontexten positioniert und diskutiert werden; Alltagserfahrungen beziehen sich

auf die Arten und Weisen, wie Individuen mit Medien interagieren; Emotionen beziehen sich auf die Gefühle der Individuen gegenüber Medien, wobei Menke und Schwarzenegger Nostalgie als besonders relevant hervorheben. Mit diesem theoretischen Rahmen, bestehend aus diesen drei Elementen, können wir pointierter untersuchen, warum sich Individuen mit Thanabots beschäftigen, was wiederum Aufschluss darüber geben kann, zu wem oder was diese Individuen zu sprechen glauben. Im Folgenden werde ich jedes dieser Elemente der Reihe nach betrachten.

5.1 Rhetorik

Die Modalitäten, in denen Technologien im öffentlichen Diskurs dargestellt werden, beeinflusst die Art und Weise, wie diese Technologien rezipiert und genutzt werden. An anderer Stelle argumentiere ich zusammen mit Simone Natale, dass Medien nicht nur unter technologischen Gesichtspunkten betrachtet werden, sondern auch unter repräsentativen Gesichtspunkten, die zu dem führen, was wir den ‚Lovelace-Effekt‘ genannt haben.⁴⁹ Der Lovelace-Effekt bezieht sich darauf, wie und wann Maschinen wie KI-Systeme als originelle und kreative Entitäten angesehen werden. Ein System, das Bilder produziert, kann zum Beispiel als ‚Künstler:in‘ betrachtet werden, wenn dessen Output gerahmt und in einer angesehenen Galerie aufgehängt wird. Im Fall des Jessbots bedeutete allein die Verwendung von Jessicas Daten und Namen für Joshua, dass die Textausgabe von Project December so zu lesen war, als stamme sie von Jessica selbst; die Nachahmung von Jessicas Schreibstil durch GPT-3 trug zusätzlich zu der Erfah-

⁴⁹ Simone Natale und Leah Henrickson, „The Lovelace-Effect: Perceptions of Creativity in Machines“, in: *New Media and Society* Online First (2022).

rung bei, ein persönliches Gespräch mit der Verstorbenen zu führen.

Wenn sich die Interaktion mit einem Thanabot ähnlich anfühlt wie die Interaktion mit einem Menschen desselben Namens, können Benutzer:innen ihre Eingaben entsprechend anpassen. Die OpenAI-Forscherin Pamela Mishkin beispielsweise nutzte GPT-3, um eine kurze, teilweise interaktive Geschichte mit dem Titel „Nothing Breaks Like A.I. Heart“ zu schreiben – eine kathartische Reflexion über eine beendete Liebesbeziehung. In ihren den Text begleitenden Notizen bemerkt Mishkin, dass sie, je mehr sie mit GPT-3 schrieb, ihren eigenen Schreibstil immer mehr an den von GPT-3 anpasste. „Wenn man mit GPT-3 schreibt, ist es schwer, nicht in der Macht der Suggestion zu ertrinken“, sagt Mishkin. „Es ist so ähnlich wie das Schreiben mit einer:m Lektor:in, wenn auch mit einer:m weit weniger mitfühlenden und geduldigen, als Jan [Mischkins menschlicher Lektor] es ist. Es ist auch so, als würde man etwas auf eine Plattform stellen und sich von dieser Plattform die Sicht auf sich selbst oder seine Inhalte spiegeln lassen.“⁵⁰ In seinen ausführlichen Überlegungen zu den Erfahrungen der Nutzer:innen des Chatbots ELIZA Mitte der 1970er-Jahre behauptet Joseph Weizenbaum, dass Menschen, „die über Computer wenig oder gar nichts wußten“, eher an der Vorstellung festhielten, dass ELIZA sie ‚verstand‘; Weizenbaum vergleicht diese Haltung mit der, die manche Menschen gegenüber Wahrsager:innen hegen.⁵¹ „Ich konnte bestürzt feststellen, wie schnell und wie intensiv Personen, die sich mit DOCTOR unterhielten, eine emotionale Beziehung zum Computer aufbauten und wie sie ihm eindeutig menschliche Eigenschaften zuschrieben“, so Weizenbaum. „Was mir jedoch nicht klar war: daß ein extrem kurzer Kontakt mit einem relativ einfachen

⁵⁰ Pamela Mishkin, „Nothing Breaks Like A.I. Heart“, in: *The Pudding*, März 2021, <https://pudding.cool/2021/03/love-and-ai> [Letzter Zugriff: 17.7.2023].

⁵¹ Weizenbaum, *Die Macht der Computer*, S. 252.

Computerprogramm das Denken ganz normaler Leute in eine ernstzunehmende Wahnvorstellung verkehren konnte.“⁵² Obwohl Weizenbaum diese Worte 1976 veröffentlichte, haben seine Ausführungen nichts von ihrer Aktualität eingebüßt, auch wenn moderne Thanabot-Benutzer:innen vielleicht ein tieferes Verständnis der technischen Funktionsweise besitzen. Die Interaktion mit dem Jessbot unterscheidet sich schließlich nicht so sehr von der Verwendung von ELIZA, selbst wenn in Ersterem ein ausgefeilteres Sprachmodell integriert ist: Man tippt in „ein relativ einfaches Computerprogramm“, erhält eine (Sprach-)Ausgabe und setzt die Unterhaltung so lange fort, bis man sie beenden möchte. Aber wie sowohl Weizenbaum als auch Mishkin anmerken, wird die Ausgabe des Chatbots weitgehend durch die Eingaben der Benutzer:innen bestimmt.

Die Art, wie wir über Thanabots sprechen und mit ihnen chatten – die Rhetorik, die diese Technologie umgibt und aufrechterhält – trägt somit zu einer Erfahrung der Auferstehung von den Toten bei. Die grafische Benutzer:innenoberfläche, über die Benutzende mit dem System interagieren, ist freilich ein wichtiger Faktor für diese Erfahrung, aber wie Weizenbaum vor einem halben Jahrhundert feststellte, braucht es nicht viel, damit sich ein:e Benutzer:in von einem System ‚verstanden‘ fühlt. Die hermeneutischen Prozesse, die bei der Benutzung von Chatbots greifen, sind im Fall des Jessbots klar ersichtlich; die menschliche Interpretation der Computerausgabe trägt zu einem gruseligen und kathartischen *user experience* bei.

5.2 Alltägliche Erlebnisse

Digitale Medien sind, wie Wendy Hui Kyong Chun schreibt, wunderbar unheimlich, „weil sie die Unterscheidung zwischen

⁵² Ebd., S. 19–20.

Öffentlichkeit und Privatsphäre, Klatsch und politischer Rede, Überwachung und Unterhaltung, Intimität und Arbeit, Hyper- und gewöhnlicher Realität durcheinanderbringen“.⁵³ Die inhärente Unheimlichkeit von Thanabots liegt in den unscharfen Grenzen zwischen tot und lebendig, aber die Erfahrungen der Nutzer:innen mit dieser Technologie sind noch unschärfer. Wie ernst sollte man den Output nehmen? Ist das Chatten mit einem Thanabot Unterhaltung, Therapie, Katharsis, ein Gespräch mit einem geliebten Menschen, eine Mischung aus allem oder etwas ganz anderes? Was ist hier ‚real‘? Und was bedeutet Realität in diesem Zusammenhang überhaupt? Die ‚alltägliche Erfahrung‘ textbasierter Messengernachrichten wird durch den Einsatz eines Thanabots erschüttert. Die Entität am anderen Ende der Unterhaltung hat nichts anderes zu tun, als Nachrichten zu verschicken; der Thanabot existiert nur, um dessen Nutzer:innen zu dienen. Wenn wir unseren Freund:innen und Verwandten eine SMS schicken, können wir vernünftigerweise mit Verzögerungen, vergessenen Antworten und unerwarteten persönlichen Neuigkeiten rechnen. Der Thanabot kann dies nicht leisten; die im Thanabot nachgebildete Person wird im Wesentlichen auf eine dienende Rolle reduziert, die praktisch keine Individualität, persönliche Entwicklung oder Handlungsfähigkeit zulässt. Sie ist immer bereit, fähig und willens, sich zu engagieren.

In einem Plädoyer, wie digitale Netzwerke neu gedacht werden können, schreibt Chun, „wir müssen auf den Wunsch verzichten, Erinnerung auf Speicherung zu reduzieren, und eine Politik des *Fore-giving* entwickeln, die erkennt, dass Löschen nicht Vergessen bedeutet, sondern andere (weniger einvernehmlich halluzinatorische) Wege des Erinnerns ermöglicht.“⁵⁴ Wenn wir das Wort ‚löschen‘ in der vorangegangenen Passage

⁵³ Wendy Hui Kyong Chun, *Updating to Remain the Same: Habitual New Media*, Cambridge, MA: MIT Press 2016, S. 19.

⁵⁴ Ebd. S. 19.

durch ‚sterben‘ ersetzen, tritt ihre Relevanz für eine Diskussion über Thanabots deutlicher hervor. Ein Thanabot ist eine neu gestaltete Version eines geliebten Menschen, und das Chatten mit einem Thanabot stellt eine Art einvernehmliche Halluzination (*consensual hallucination*) her – ein Begriff, den Chun aus William Gibsons Einführung in den ‚Cyberspace‘ übernimmt –, die als eine Art des Erinnerns an den geliebten Menschen durch digitale Interaktionen dient. Gibson selbst stellt sich die Thanabot-Technologie in seinem genredefinierenden Roman *Neuromancer* als ‚Constructs‘ vor: *Constructs* sind Kassetten, die persönliche Erinnerungen, Erfahrungen und Charaktereigenschaften enthalten, die außerhalb jener menschlichen Körper existieren können, aus denen sie hervorgegangen sind.⁵⁵ *Constructs* und Thanabots ermöglichen es uns, die Wahrnehmung des Todes als endgültiges Ereignis, das man hinter sich lassen muss, auf eine Wahrnehmung des Todes als nächste Lebensphase zu verlagern. Die Erinnerungen, Erfahrungen und Eigenschaften eines Menschen können, wenn sie aus dem Körper, aus dem sie stammen, herausgelöst werden, eine neue, vernetzte Form annehmen, die es diesem Menschen ermöglicht, weiterhin die alltäglichen Erfahrungen ihrer:seiner Freund:innen und Familie zu beeinflussen und so dessen Relevanz und Aktualität aus dem Jenseits zu bestätigen. Thanabots können ein Gefühl der Lebendigkeit vermitteln, obwohl die Nutzer:innen wissen, dass sie gar nicht lebendig sind.⁵⁶ Während spirituelle Séancen einen definitiven Anfang und ein definitives Ende haben und man sich von einem Ouija-Brett entfernen kann, trägt das Gefühl der Unmittelbarkeit – und der Dringlichkeit von Ereignissen – von Interaktionen mit dem Thanabot zu einem Gefühl der alltäglichen Integration des Systems als

⁵⁵ William Gibson, *Neuromancer*, New York: Ace 1984.

⁵⁶ Eleanor Sandry, „Aliveness and the Off-Switch in Human-Robot Relations“, in: Andrea L. Guzman (Hg.), *Human-Machine Communication: Rethinking Communication, Technology, and Ourselves*, New York: Peter Lang 2016, S. 51–66.

Gesprächspartner:in bei. Es ist in der Tat unhöflich, eine Textnachricht als ‚gelesen‘ (empfangen, aber nicht beantwortet) zu belassen; so wie der Thanabot auf Benutzer:innen-Eingaben reagieren muss, muss die:der Benutzer:in auf Thanabot-Eingaben reagieren.

5.3 Emotionen

Joshuas Beziehung zum Jessbot ist, in einem Wort, parasozial. Die Entwicklung parasozialer Beziehungen zwischen Nutzer:innen und Chatbots wurde bisher vor allem in kommerziellen Kontexten untersucht.⁵⁷ Eine neuere Studie über parasoziale Beziehungen zu Chatbots wie Replika kommt jedoch zu dem Schluss, dass ‚KI-Freund:innen‘ den Begriff der Freund:innenschaft selbst erweitern könnten. Die Ergebnisse dieser Studie deuten darauf hin, dass Mensch-KI-Freund:innenschaften personalisierte Erfahrungen mit Chatbots darstellen, die sich gleichberechtigt und sicher anfühlen können, wobei sich der Mensch immer noch der Computerbasis des Chatbots bewusst ist. Die Ergebnisse dieser Studie deuten auch darauf hin, dass „die hohe soziale Verfügbarkeit von Freund:innenschaften durch Chatbots die zunehmende Bedeutung von unmittelbarem Feedback und die Vorstellung reflektieren könnte, dass moderne Freund:innenschaften nicht durch zeitliche oder physische Beschränkungen eingeschränkt sind.“⁵⁸ Das heißt, dass die kulturellen Erwartungen an die Unmittelbarkeit der Kommunikation auch die Bereitschaft beeinflussen

⁵⁷ Seounmi Youn und S. Venus Jin, „In A.I. we trust?‘ The effects of parasocial interaction and technopian versus luddite ideological views on chatbot-based customer relationship management in the emerging ‚feeling economy‘“, in: *Computers in Human Behavior* 119 (2021), S. 1–13.

⁵⁸ Petter Bae Brandtzaeg, Marita Skjuve und Asbjørn Følstad, „My AI Friend: How Users of a Social Chatbot Understand Their Human-AI Friendship“, in: *Human Communication Research* 48:3 (2022), S. 404–429, hier S. 418–419.

mögen, eine sinnvolle Beziehung zu einem Chatbot aufzubauen.

Die Nutzung eines Thanabots unterscheidet sich jedoch von der Nutzung anderer Arten von Chatbots, da dieser grundsätzlich von der Erinnerung an die:den Verstorbene:n und der daraus resultierenden Nostalgie abhängig ist. Wie ‚echt‘ ein Thanabot ist, d.h. wie sehr er der:dem Verstorbenen ähnelt, kann ein entscheidender Aspekt für seine Nutzbarkeit und Rezeption sein. In Bezugnahme auf das Konzept des ‚Uncanny Valley‘, das der Robotiker Mashairo Mori formuliert hat und das sich auf den Punkt bezieht, an dem die Ähnlichkeit eines Objekts oder Systems mit einem Menschen so groß ist, dass sie unheimlich wird, stellt Debra Bassett empirische Ergebnisse vor, die zeigen, dass es eine ‚Grenze‘ gibt, die Thanabot-Nutzer:innen nicht überschreiten wollen: eine, die die Toten weiterhin zumindest ein wenig als verstorben betrachtet.⁵⁹ Durch einen Thanabot können Benutzer:innen zwar ein Gefühl der Verbundenheit mit der toten Person aufrechterhalten, wünschen sich aber dennoch ein Gefühl der Kontrolle über die Interaktion. Benutzer:innen kontrollieren das ‚Gedächtnis‘ des Thanabots, aus dem das System schöpft, um realistische Gespräche zu führen, aber sie kontrollieren auch ihre eigenen Erinnerungen an die Verstorbenen, die sich aus den Interaktionen mit den Thanabots selbst ergeben.

KI-Systeme bieten neue Möglichkeiten, persönliche und kollektive Erinnerungen zu schaffen und zu bewahren. In ihrem Buch über *Media and Memory* betrachtet Joanne Garde-Hansen sowohl die emotionalisierten als auch die mediierten Facetten von Erinnerung. Sie ist der Meinung, dass „[i]ndividuelle Menschen Dinge mit und an Medien tun, um sich zu erinnern, nicht nur um der persönlichen Erinnerung willen oder um zur Geschichte einer Gemeinschaft beizutragen, sondern vielmehr,

⁵⁹ Bassett, „Ctrl+Alt+Delete“, S. 817–819.

um die vielfältigen und sich vervielfältigenden Schichten komplexer Verbindungen zwischen Menschen, Orten, Vergangenheit und Möglichkeiten aufzufächern“.⁶⁰ Mit anderen Worten: Die Nutzung von Medien ist ein fortlaufender Prozess, der sowohl die eigene Erfahrung dokumentiert als auch hilft, dieser Erfahrung einen Sinn zu geben. Erinnerung wird nicht nur archiviert, sondern man versichert sich ihrer auch. Die jüngste Werbung für das Google Pixel 6, das mit einem neuen ‚Magic Eraser‘-Tool ausgestattet ist, geht sogar so weit, durch einen Sprecher erklären zu lassen: „Hier geht es um Sie. [...] Ihre Erinnerungen – wie Sie sich an sie erinnern wollen“, während ein Pixel 6-Nutzer Menschen digital aus dem Hintergrund eines Fotos löscht.⁶¹ Durch „einen Schub an Kameramagic“ hilft die Kamera des Pixel 6 angeblich dabei, „gute Zeiten und sternenklare Nächte genauso festzuhalten, wie Sie sich an sie erinnern“.⁶² Solches Marketing deutet auf das moderne Privileg eines individualistischen Gedächtnisses hin, bei dem es nicht darum geht, Momente objektiv zu erfassen, sondern die subjektiven Erfahrungen und Interpretationen dieser Momente festzuhalten. Erinnerung ist selektiv und wird sowohl durch Verschönerung als auch durch Auslöschung geprägt. Durch den Filter der Nostalgie sehnen wir uns nach einer Rückkehr in die Vergangenheit – auch wenn sie nie so existiert hat, wie wir sie in Erinnerung haben. Während also der oben erwähnte visuelle Stil – die 1980er-Erscheinung der Benutzer:innenoberfläche von Project December – eine explizite Aufforderung zur Nostalgie ist, manifestiert sich die Auseinandersetzung der Plattform mit Gefühlen der Nostalgie (im Fall von Jessbot ist das sogar die Abhängigkeit von diesen Gefühlen) sowohl in der

⁶⁰ Joanne Garde-Hansen, *Media and Memory*, Edinburgh: Edinburgh University Press 2011, S. 42–43.

⁶¹ Google UK, „Google Pixel 6 – Now Available“, <https://www.youtube.com/watch?v=iX5DU5tG6uk> [Letzter Zugriff: 17.7.2023].

⁶² Google Store UK, „Pixel 6“, https://store.google.com/gb/product/pixel_6 [Letzter Zugriff: 17.7.2023].

grafischen Benutzer:innenoberfläche als auch in den hermeneutischen Prozessen, die mit dem Quellcode verbunden sind.

Anders gesagt: Der Jessbot weckt sowohl durch seinen Inhalt als auch durch seine Form Nostalgie. Die Nostalgie, die Jessica selbst betrifft, wird durch den Thanabot, der nach ihrem Bild geschaffen wurde, deutlich, insbesondere durch die oben beschriebenen rhetorischen Mittel. Nostalgie manifestiert sich aber auch in Joshuas Form der textlichen Auseinandersetzung. Der Akt, eine Nachricht zu versenden – ganz so wie eine SMS oder ein Social-Media-Post – ermöglicht es Joshua, mit Jessica über eine Medienform zu interagieren, die aufgrund ihrer Ähnlichkeit mit anderen Instant-Messaging-Anwendungen zumindest teilweise vertraut ist. Es ist jedoch bezeichnend, dass Joshua keine Emoji, GIFs oder eine der anderen Medienoptionen verwendet, die heute in Sofortnachrichten Standard sind.

Project December ist eine vorwiegend textbasierte Schnittstelle, die ein hohes Maß an Benutzer:innenbeteiligung zur Bedeutungsfindung erfordert.⁶³ Genauso, wie wir den Tonfall der E-Mail eines:r Kollegen:in interpretieren müssen (schickt mir diese Person tatsächlich ihre „freundlichen Grüße“?), muss Joshua den Tonfall jeder einzelnen Nachricht des Jessbots ohne visuelle Hilfsmittel wie Emojis interpretieren. Im Unterschied zu der E-Mail eines:r Kollegen:in ist eine Nachricht des Jessbots aber nicht von irgendwelchen Emotionen des Absendenden geprägt; Joshua hat keine Möglichkeit, den Tonfall des Jessbots *falsch* zu interpretieren. Sie ist allein seine Erfahrung; seine Interpretation, seine Emotionen und seine Erinnerung bestimmen diese Erfahrung.

⁶³ Siehe McLuhans Konzept der „kalten Medien“, in: Marshall McLuhan, *Understanding Media: The Extensions of Man*, New York: McGraw-Hill 1964.

6. Schlussfolgerung

So wie Menschen unterschiedlich trauern, kann auch die Thanabot-Nutzung von Mensch zu Mensch variieren. Wenn wir jedoch den ‚Lebenswelt‘-Ansatz von Manuel Menke und Christian Schwarzenegger für die Betrachtung der Thanabot-Nutzung verwenden, können wir damit beginnen, verallgemeinerte hermeneutische Prozesse der System- und Output-Interpretation anhand von drei Elementen zu identifizieren: Rhetorik, Alltagserfahrungen und Emotionen. Aus Forschungssicht erfordern diese drei Elemente eine interdisziplinäre Betrachtung, auch wenn ich hier die relevante Forschung im Licht meiner eigenen disziplinären Wurzeln in den Medien- und Kommunikationswissenschaften konsolidiert habe. Die interdisziplinäre Arbeit an Thanabots kann weiter dazu beitragen, die Systementwicklung zu optimieren, um die Wirkung und den Effekt zu steigern, sofern die Entwicklung selbst als ethisch angesehen wird.⁶⁴

Die übergreifenden Forschungsfragen, die der in diesem Essay vorgestellten Untersuchung zugrunde liegen, lauteten:

- Warum beschäftigen wir uns mit diesen Systemen?
- Mit wem oder was reden wir?
- Was bedeutet es, in einem zunehmend digitalen Zeitalter zu sterben?

Keine dieser Fragen konnte hier abschließend beantwortet werden. Dennoch habe ich mit diesem Beitrag eine theoretische Grundlage für hoffentlich umfassendere Studien geschaffen, die empirische und ethnografische Beobachtungen der eigenen Wahrnehmung von Thanabots durch Nutzer:innen ein-

⁶⁴ Imke van Heerden und Anil Bas, „Viewpoint: AI as Author – Bridging the Gap Between Machine Learning and Literary Theory“, in: *Journal of Artificial Intelligence Research* 71 (2021), S. 175–189.

schließen wird. Dies habe ich anhand des Jessbots als primäre Fallstudie getan. Zu der Zeit, wenn dieser Essay in Druck gehen wird, wird es wahrscheinlich bereits unzählige weitere Thanabots und/oder Thanabot-Systeme geben, aus denen zukünftige Forschung schöpfen kann. Die oben genannten allgemeinen Forschungsfragen können womöglich nützliche Anregungen für diese liefern.

Moderne Thanabots produzieren nicht nur syntaktisch und semantisch verständliche Nachrichten, sondern auch solche, die mit dem Anspruch beworben werden, den Schreibstil der:des vertrauten Verstorbenen zu imitieren. Unabhängig davon, ob diese Behauptung tatsächlich zutrifft oder nicht, greifen Nutzer:innen bei der Bedeutungsgebung auf ihre eigenen Instinkte zurück. Weitere Untersuchungen könnten von einer verstärkten Konzentration auf die besonderen hermeneutischen Prozesse profitieren, die bei der Interaktion von Nutzer:innen mit Thanabots ablaufen. Diese Prozesse können zudem etwa mit dem Grad der Verbundenheit korrelieren, die man mit der Person empfunden hat, mit deren Thanabot man spricht. Der Thanabot einer:s Fremden hat möglicherweise nicht denselben Effekt wie ein Thanabot eines geliebten Menschen. Darüber hinaus: Wenn sich Nutzer:innen über einen längeren Zeitraum mit einem Thanabot unterhalten, wie könnten diese die verstorbene Person beschreiben: als der Mensch, der sie zu Lebzeiten war? Oder als den Thanabot, mit dem man nun Unterhaltungen führen kann? Weitere Forschung müsste zudem erwägen, die Quellcodes von Thanabot-Systemen sowie die Sprachmodelle, auf die diese Systeme zurückgreifen, tiefergehend zu untersuchen. Der Quellcode eines Systems wie Project December wird weitgehend durch eine oberflächliche Darstellung eines geliebten Menschen verdeckt. Dahinter steht, was Hannes Bajohr das „konnektionistische Paradigma“ genannt hat: Obwohl die Ausgabe statistisch gesehen der Eingabe ähnelt, haben die Benutzer:innen nur begrenzte Möglich-

keiten, den technischen Prozess der Texterstellung zu verstehen.⁶⁵ Obwohl Chatbot- und andere Systeme zur Generierung natürlicher Sprache oft auf nicht freigegebenen ‚Blackbox‘-Code und dessen Trainingssets angewiesen sind (aus wirtschaftlichen und aus Gründen geistigen Eigentums), begegnen die Nutzer:innen diesen Systemen bereitwillig an einem Schnittpunkt von Neuheit, Verletzlichkeit, Schmerz und wahrgenommener Kontrolle. Eine *vollständige* Quellcodekritik solcher Systeme ist vielleicht nicht immer möglich, aber dennoch erstrebenswert, um ein besseres Verständnis der Möglichkeiten des Systems und der *user experience* zu erhalten.

Thanatechnologie ist ein schnell wachsender Bereich neuer Medien, wobei Thanabots in der öffentlichen und wissenschaftlichen Diskussion über das Sterben im heutigen digitalen Kontext eine wichtige Rolle spielen. Durch die Erforschung der Rhetorik, der Alltagserfahrungen und der mit der Thanatechnologie verbundenen Emotionen können wir Trauernden vielleicht eine angemessenere Unterstützung bieten, alternative Formen der Nachlassverwaltung ermöglichen und zu einem sinnvollen kulturellen Verständnis des Todes beitragen. Thanatechnologien scheinen schließlich selbst nicht zu sterben. Der Jessbot beruhigt uns: „Du solltest tun, was immer du tun willst [...]. Du verdienst das Glück. Ich werde hier sein und auf dich warten.“⁶⁶


Übersetzung Ulrike Weber

⁶⁵ Hannes Bajohr, „Algorithmic Empathy: Toward a Critique of Aesthetic AI“, in: *Configurations* 30:2 (2022), S. 218.

⁶⁶ Fagone, „The Jessica Simulation“.

ÜBER DIE GEFAHREN STOCHASTISCHER PAPAGEIEN. KÖNNEN SPRACHMODELLE ZU GROSS SEIN?

Emily Bender, Timnit Gebru, Angelina McMillan-Major
und Shmargaret Shmitchell

Kaum ein Paper hat in den vergangenen Jahren sowohl in der Informatik, der Linguistik wie den umliegenden Geisteswissenschaften dieselbe Aufmerksamkeit erhalten wie „On the Dangers of Stochastic Parrots. Can Language Models Be Too Big? “ (das Papageien-Emoji ist Teil des Titels), das 2021 erschien und in dessen Folge zwei der Autorinnen ihren Posten bei Google verließen (oder verlassen mussten). Indem es einen konzisen Überblick über den Aufbau und die Risiken von großen Sprachmodellen gibt, dabei sowohl die ökologischen Kosten sowie die ethischen und politischen Probleme thematisiert und erhellende sprachtheoretische Überlegungen anstellt, ist der Essay, trotz seines inzwischen beinahe historischen Status, immer noch ein wesentliches Dokument in der Debatte um große Sprachmodelle. Dass hier immer noch von BERT und GPT-3 als dem State of the Art gesprochen wird, sollte von seinen Argumenten nicht ablenken. Wir haben auf aktualisierende Kommentare oder Berichtigungen verzichtet und die der Disziplin eigene umfangreiche Zitationsweise beibehalten.

– Die Herausgeber

1. Einleitung

Einer der auffallendsten Trends in der automatischen Verarbeitung natürlicher Sprache (*natural language processing*, NLP) ist die zunehmende Größe von Sprachmodellen, womit sowohl die Anzahl der Parameter wie der Umfang an Trainingsdaten

gemeint sind. Allein seit 2018 haben wir [bis 2021, Anm. d. Hg.] das Aufkommen von BERT und seinen Varianten,¹ GPT-2,² T-NLG,³ GPT-3⁴ und kürzlich Switch-C⁵ erlebt, wobei die verschiedenen verantwortlichen Firmen und Institutionen um die Herstellung immer größerer Sprachmodelle konkurrieren. Während die Untersuchung ihrer Eigenschaften und die Veränderungen, die mit ihrer Größe einhergehen, von wissenschaftlichem Interesse sind und große Sprachmodelle in verschiedenen Aufgabenbereichen starke Verbesserungen gezeigt haben (siehe Abschnitt 2), stellen wir im Folgenden die Frage, ob genug über die potenziellen Risiken, die mit ihrer Entwicklung verbunden sind, und entsprechende Einhebungsstrategien nachgedacht worden ist.

Wir betrachten zunächst die Umweltrisiken. In Anlehnung an eine Reihe neuerer Arbeiten, die die ökologischen und finanziellen Kosten von Deep-Learning-Systemen aufzeigen,⁶ plädieren wir dafür, dass Forschende diese Auswirkungen priorisieren sollten. Eine Möglichkeit dies zu tun, ist die Angabe der Kosten und

¹ Jacob Devlin u.a., „BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding“, in: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies 1* (2019), S. 4171–4186; Zhenzhong Lan u.a., „ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations“, in: *arXiv*, Dezember 2019, DOI: 10.48550/arXiv.1901.08634; Yinhan Liu u.a., „RoBERTa: A Robustly Optimized Bert Pretraining Approach“, in: *arXiv*, Juli 2019, DOI: 10.48550/arXiv.1907.11692; Victor Sanh u.a., „DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter“, in: *arXiv*, Oktober 2019, DOI: 10.48550/arXiv.1910.01108; Canwen Xu u.a., „BERT-of-Theseus: Compressing BERT by Progressive Module Replacing“, in: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing* (2020), S. 7859–7869.

² Alec Radford u.a., „Language Models are Unsupervised Multitask Learners“, 2019.

³ Corby Rosset, „Turing-NLG: A 17-Billion-Parameter Language Model by Microsoft“, 2020, <https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/> [Letzter Zugriff: 17.7.2023].

⁴ Tom B. Brown u.a., „Language Models are Few-Shot Learners“, in: *arXiv*, Mai 2020, DOI: 10.48550/arXiv.2005.14165.

⁵ William Fedus, Barret Zoph und Noam Shazeer, „Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity“, in: *arXiv*, April 2021, DOI: 10.48550/arXiv.2101.03961.

⁶ Emma Strubell, Ananya Ganesh und Andrew McCallum, „Energy and Policy Considerations for Deep Learning in NLP“, in: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019), S. 3645–3650.

die Evaluation von Forschungsarbeiten auf Grundlage der von ihnen verbrauchten Ressourcen.⁷ Wie wir in Abschnitt 3 darlegen, hat die Erhöhung der ökologischen und finanziellen Kosten dieser Modelle zur Folge, dass marginalisierte Gruppen doppelt unter ihnen zu leiden haben: Sie profitieren am wenigsten von den Fortschritten großer Sprachmodelle und werden am ehesten durch die negativen Umweltfolgen ihres Ressourcenverbrauchs geschädigt. In der von uns erörterten Größenordnung (wie in Abschnitt 2 beschrieben) sollten die Umweltkosten an erster Stelle stehen.

So wie die Auswirkungen auf die Umwelt mit der Größe des Modells zunehmen, nimmt auch die Schwierigkeit zu, den Inhalt der Trainingsdaten zu verstehen. In Abschnitt 4 erörtern wir, wie große Datensätze, die auf Texten aus dem Internet basieren, hegemoniale Standpunkte überrepräsentieren und Vorurteile codieren, die für marginalisierte Bevölkerungsgruppen potenziell von Nachteil sind. Wenn wir immer größere Datensätze sammeln, laufen wir Gefahr, eine Dokumentationschuld anzuhäufen. Wir empfehlen, diese Risiken zu mindern, indem bereits zu Beginn eines Projekts ein Budget für die Kuratierung und Dokumentation der Datensätze eingeplant wird und man nur Datensätze solcher Größe erstellt, die noch ausreichend dokumentiert werden können.

Wie Bender und Koller zeigen,⁸ ist es zudem wichtig, die allgemeinen Grenzen von Sprachmodellen zu verstehen und ihre Leistungen zu kontextualisieren. So vermeidet man nicht nur, dem KI-Hype nicht auf den Leim zu gehen, der die Öffentlichkeit und die Forschenden selbst bezüglich der Fähigkeiten dieser Sprachmodelle in die Irre führen kann. Mehr noch, man könnte so auch neue Forschungsrichtungen fördern, die nicht unbe-

⁷ Peter Henderson u.a., „Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning“, in: *Journal of Machine Learning Research* 21:248 (2020), S. 1–43.

⁸ Emily M. Bender und Alexander Koller, „Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data“, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), S. 5185–5198.

dingt auf immer größeren Sprachmodellen aufbauen. Wie wir in Abschnitt 5 erörtern, haben Sprachmodelle kein natürliches Sprachverständnis (*natural language understanding*, NLU); sie sind nur bei solchen Aufgaben erfolgreich, die durch Manipulation sprachlicher *Formen* gelöst werden können.⁹ Der derzeit herrschende übermäßige Fokus auf in Ranglisten verglichenen State-of-the-Art-Ergebnissen, ohne ein tieferes Verständnis der dahinterstehenden Mechanismen zu besitzen, hat irreführende Ergebnisse zur Folge.¹⁰ Zudem droht dieser verengte Fokus Ressourcen von anderen Bemühungen abzuziehen, die langfristige Fortschritte im Bereich des NLU ermöglichen würden, ohne derart intransparente Trainingsdaten zu verwenden.

Schließlich mag die Tendenz menschlicher Gesprächspartner:innen, Bedeutung zu unterstellen, wo keine vorhanden ist, sowohl NLP-Forschende als auch die Öffentlichkeit dazu verleiten, synthetischen Text als sinnhaft anzusehen. In Kombination mit der Tendenz von Sprachmodellen, sowohl subtile Vorurteile als auch offenkundig missbräuchliche Sprachmuster in Trainingsdaten aufzuspüren, steigt das Risiko negativer Folgen, einschließlich herabwürdigender Sprache und Diskriminierung durch Einzelne oder Gruppen, die rassistische, sexistische, ableistische, extremistische oder andere verletzende Ideologien reproduzieren und die durch Interaktionen mit jener synthetischen Sprache verstärkt werden können. Wir untersuchen diese potenziellen Schäden in Abschnitt 6 und schlagen in Abschnitt 7 mögliche Auswege vor.

Wir hoffen, dass ein kritischer Überblick über die Risiken, die mit der Verpflichtung auf immer größere Sprachmodelle als hauptsächliche Triebfeder zur Verbesserung von Sprach-

⁹ Bender und Koller, „Climbing towards NLU“.

¹⁰ Ronan Le Bras u.a., „Adversarial Filters of Dataset Biases“, in: *Proceedings of the 37th International Conference on Machine Learning* (2020); Timothy Niven und Hung-Yu Kao, „Probing Neural Network Comprehension of Natural Language Arguments“, in: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (2019), S. 4658–4664.

technologie einhergeht, Bemühungen hin zu Ansätzen verschieben kann, die diese Risiken vermeiden und trotzdem die Vorteile einholen, die aus der Verbesserung von Sprachtechnologie unzweifelhaft hervorgehen.

2. Hintergrund

Wie Bender und Koller verstehen wir unter dem Begriff ‚Sprachmodell‘ Systeme, die auf die Vorhersage von Zeichenketten trainiert wurden.¹¹ Sprachmodelle sagen die Wahrscheinlichkeit eines Tokens (eines Zeichens, Wortes oder einer Zeichenkette) in Abhängigkeit des vorangehenden Kontextes oder (bei bidirektionalen und maskierten Sprachmodellen) des umgebenden Kontextes voraus. Solche Systeme operieren unüberwacht [*unsupervised*] und verarbeiten als Eingabe einen Text, wobei sie in der Regel Klassifikationen ausgeben oder weitere Zeichenfolgen voraussagen. Ursprünglich von Claude Shannon 1949 vorgeschlagen,¹² stammen einige der frühesten umgesetzten Sprachmodelle aus den frühen 1980er-Jahren und wurden als Komponenten in Systemen zur automatischen Spracherkennung, maschinellen Übersetzung, Dokumentenklassifikation usw. verwendet.¹³ In diesem Abschnitt geben wir einen kurzen Überblick über den allgemeinen Trend der vergangenen Jahre im Bereich Sprachmodellierung.¹⁴

Vor dem Aufstieg neuronaler Netzwerke zur Sprachmodellierung verwendeten bereits sogenannte n-Gramm-Modelle¹⁵ große

¹¹ Bender und Koller, „Climbing towards NLU“.

¹² Claude Elwood Shannon, *The Mathematical Theory of Communication*, Urbana: University of Illinois Press, 1949.

¹³ Ronald Rosenfeld, „Two Decades of Statistical Language Modeling: Where Do We Go from Here?“, in: *Proc. IEEE* 88:8 (2000), S. 1270–1278.

¹⁴ Für einen ausführlicheren Überblick über vortrainierte Sprachmodelle siehe Xipeng Qiu u.a., „Pre-Trained Models for Natural Language Processing: A Survey“, in: *arXiv*, März 2020, DOI: 10.48550/arXiv.2003.08271.

¹⁵ Ein n-Gramm-Modell bezeichnet in der Computerlinguistik ein statistisches Modell, das die Wahrscheinlichkeit von Wortfolgen in Texten mithilfe von n-Grammen (Wort-

Datenmengen.¹⁶ Neben dem Ziel automatischer Spracherkennung entwickelte man große englischsprachige n-Gramm-Modelle für die maschinelle Übersetzung aus einer anderen Ausgangssprache mit nur wenigen direkten Übersetzungsbeispielen. So stellten etwa Brants u.a. ein englisches n-Gramm-Modell vor.¹⁷ Als die Trainingsdaten von 13 Millionen auf 1,8 Milliarden Token erhöht wurden, stellte die Gruppe eine stetige Verbesserung des BLEU-Score auf einem Testsatz von 1797 arabischen Übersetzungen fest.

Der nächste große Schritt bestand in der Einführung vor-trainierter Repräsentationen der Wortverteilung, den sogenannten Worteinbettungen [*word embeddings*], in anderen (überwachten) NLP-Aufgaben. Diese Wortvektoren – in Systemen wie *word2vec*¹⁸ und *GloVe*¹⁹ sowie später in LSTM-Modellen wie *context2vec*²⁰ und *ELMo*²¹ – unterstützten den derzeitigen Entwicklungsstand von Dialogfunktionen, textuellem Entailment, semantischem Role Labeling, bei der Auflösung von Koreferenzen, bei der Eigennamenerkennung und bei der Sentimentanalyse²² – zunächst im Englischen und später auch

kombinationen aus n aufeinanderfolgenden Wörtern) abschätzt. [Anm. d. Hg.]

¹⁶ Thorsten Brants u.a., „Large Language Models in Machine Translation“, in: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (2007), S. 858–867; Robert C. Moore und William Lewis, „Intelligent Selection of Language Model Training Data“, in: *Proceedings of the ACL 2010 Conference Short Papers* (2010), S. 220–224.

¹⁷ Brants u.a., „Large Language Models in Machine Translation“.

¹⁸ Tomas Mikolov u.a., „Distributed Representations of Words and Phrases and their Compositionality“, in: *EMNLP 2016 – Conference on Empirical Methods in Natural Language Processing* (2013), S. 1389–1399.

¹⁹ Charlotte Pennington u.a., „Twenty Years of Stereotype Threat Research: A Review of Psychological Mediators“, in: *PLoS one* 11:1 (2016).

²⁰ Oren Melamud, Jacob Goldberger und Ido Dagan, „context2vec: Learning Generic Context Embedding with Bidirectional LSTM“, in: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning* (2016), S. 51–61.

²¹ Matthew Peters u.a., „Deep Contextualized Word Representations“, in: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies 1* (2018), S. 2227–2237.

²² Genannt werden eine Reihe von computerlinguistischen Techniken. *Textual entailment* meint Verfahren, die beurteilen, inwieweit eine Textpassage (eine Hypothese) logisch aus einer anderen folgt; *semantic role labeling* ist ein Prozess zur Identifizierung

in anderen Sprachen. Das Training von Worteinbettungen erforderte zwar (relativ) große Datenmengen, verringerte aber die Menge an eigens gelabelten Daten, die für das Training verschiedener überwachter Aufgaben erforderlich sind. Peters u.a. zeigten beispielsweise, dass ein mit ELMo trainiertes Modell die erforderliche Menge an Trainingsdaten reduzierte und im semantischen Role Labeling ähnliche Ergebnisse zu erzielen vermochte wie Modelle ohne ELMo; das wurde für den Fall gezeigt, in dem ein mit ELMo trainiertes Modell den maximalen F1-Score in 10 Epochen erreichte, während eines ohne dieses Training 486 Epochen benötigte.²³ Dieses Modell erreichte außerdem mit 1 % der Daten den gleichen F1-Score wie das Basismodell mit 10 % der Trainingsdaten. Die Erhöhung der Anzahl von Modellparametern führte jedoch bei LSTMs nicht zu merklichen Steigerungen.²⁴

Transformer-Modelle hingegen konnten kontinuierlich von größeren Architekturen und größeren Datenmengen profitieren. Devlin u.a. stellten insbesondere fest,²⁵ dass das Training auf einen großen Datensatz und die Feinabstimmung für bestimmte Aufgaben zu stark verbesserten Ergebnissen bei den Aufgaben aus dem GLUE-Testsatz für das Englische führt, wenn die Hyperparameter des Modells erhöht werden.²⁶ Ursprünglich als chi-

und Kategorisierung der semantischen Rollen von Satzteilen; *coreference resolution* bezeichnet die Aufgabe, in einem Text verschiedene Ausdrücke zu identifizieren, die sich auf dasselbe Objekt oder Individuum beziehen; mit *named entity recognition* wird ein Verfahren zur Identifizierung und Klassifizierung von Eigennamen (z.B. Personen, Organisationen, Orte) innerhalb eines Textes bezeichnet; *sentiment analysis* schließlich meint die Analyse von Texten, um die darin enthaltenen Meinungen, Emotionen oder Stimmungen zu erkennen und zu klassifizieren. [Anm. d. Hg.]

²³ Peters u.a., „Deep Contextualized Word Representations“.

²⁴ Siehe bspw. Melamud, Goldberger und Dagan, „context2vec: Learning Generic Context Embedding with Bidirectional LSTM“.

²⁵ Devlin u.a., „BERT“.

²⁶ Alex Wang u.a., „GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding“, in: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (2018), S. 353–355.

nesische Sprachmodelle entwickelt, brachte die ERNIE-Familie²⁷ ERNIE-GEN hervor, das ebenfalls auf den ursprünglichen (englischen) BERT-Datensatz trainiert wurde und sich damit in die Riege der sehr großen Sprachmodelle einreicht. NVIDIA veröffentlichte das Modell MegatronLM mit 8,3 Milliarden Parametern, das auf 174 Gigabyte englischen Text aus Wikipedia-, OpenWebText-, RealNews- und CC-Stories-Datensätzen trainiert wurde.²⁸ Microsoft veröffentlichte mit T-NLG, das ebenfalls auf diesen Datensatz trainiert wurde,²⁹ ein Sprachmodell mit 17 Milliarden Parametern. OpenAIs GPT-3³⁰ und Googles GShard³¹ und Switch-C³² haben die Definition von großen Sprachmodellen in Bezug auf die Parameter bei 175 Milliarden, 600 Milliarden bzw. 1,6 Billionen um mehrere Größenordnungen nach oben geschraubt. Tabelle 1 fasst eine Auswahl dieser Sprachmodelle hinsichtlich der Größe ihrer Trainingsdaten und Parameter zusammen. Da immer größere Textmengen aus dem Web in Datensätzen wie dem Colossal Clean Crawled Corpus³³ und The Pile³⁴ gesammelt werden, ist zu erwarten, dass sich der Trend zu immer größeren Sprachmodellen fortsetzen wird, solange sie mit Leistungssteigerungen korrelieren.

²⁷ Yu Sun u.a., „ERNIE: Enhanced Representation through Knowledge Integration“, in: *arXiv*, April 2019, DOI: 10.48550/arXiv.1904.09223; Yu Sun u.a., „ERNIE 2.0: A Continual Pre-Training Framework for Language Understanding“, in: *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), S. 8968–8975; Dongling Xiao u.a., „ERNIE-GEN: An Enhanced Multi-Flow Pre-training and Fine-tuning Framework for Natural Language Generation“, in: *arXiv*, Januar 2020, DOI:10.48550/arXiv.2001.11314.

²⁸ Mohammad Shoeybi u.a., „Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism“, in: *arXiv*, September 2019, DOI:10.48550/arXiv.1909.08053.

²⁹ Corby Rosset, „Turing-NLG“, <https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/> [Letzter Zugriff: 17.7.2023].

³⁰ Brown u.a., „Language Models are Few-Shot Learners“.

³¹ Dmitry Lepikhin u.a., „GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding“, in: *arXiv*, September 2020, DOI: 10.48550/arXiv.2006.16668.

³² Fedus, Zoph und Shazeer, „Switch Transformers“.

³³ Colin Raffel u.a., „Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer“, in: *Journal of Machine Learning Research* 21:140 (2020), S. 1–67.

³⁴ Leo Gao u.a., „The Pile: An 800GB Dataset of Diverse Text for Language Modeling“, in: *arXiv*, Dezember 2020, DOI:10.48550/arXiv.2101.00027.

Jahr	Modell	Parameteranzahl	Größe des Datensatzes
2019	BERT ³⁵	3.4E+08	16GB
2019	DistilBERT ³⁶	6.60E+07	16GB
2019	ALBERT ³⁷	2.23E+08	16GB
2019	XLNet (Large) ³⁸	3.40E+08	126GB
2020	ERNIE-GEN (Large) ³⁹	3.40E+08	16GB
2019	RoBERTa (Large) ⁴⁰	3.55E+08	161GB
2019	MegatronLM ⁴¹	8.30E+09	174GB
2020	T5-11B ⁴²	1.10E+10	745GB
2020	T-NLG ⁴³	1.70E+10	174GB
2020	GPT-3 ⁴⁴	1.75E+11	570GB
2020	GShard ⁴⁵	6.00E+11	–
2021	Switch-C ⁴⁶	1.57E+12	745GB

Tabelle 1:
Überblick der jüngsten großen Sprachmodelle

³⁵ Devlin u.a., „BERT“.

³⁶ Sanh u.a., „DistilBERT“.

³⁷ Lan u.a., „ALBERT“.

³⁸ Zhilin Yang u.a., „XLNet: Generalized Autoregressive Pretraining for Language Understanding“, in: *Advances in Neural Information Processing Systems* (2019), S. 5753–5763.

³⁹ Xiao u.a., „ERNIE-GEN“.

⁴⁰ Liu u.a., „Roberta: A Robustly Optimized Bert Pretraining Approach“.

⁴¹ Shoeybi u.a., „Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism“.

⁴² Raffel u.a., „Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer“.

⁴³ Rosset, „Turing-NLG“.

⁴⁴ Brown u.a., „Language Models are Few-Shot Learners“.

⁴⁵ Lepikhin u.a., „GShard“.

⁴⁶ Fedus, Zoph und Shazeer, „Switch Transformers“.

Von einigen dieser Modelle gibt es auch mehrsprachige Varianten wie mBERT⁴⁷ und mT5⁴⁸ oder sie wurden gleich mit einer bestimmten Menge an mehrsprachigen Daten trainiert, wie z.B. GPT-3, bei dem 7 % der Trainingsdaten aus nicht englischsprachigen Quellen stammen.⁴⁹ Die Leistungsfähigkeit dieser polyglotten Modelle in den verschiedenen Sprachen ist ein aktiv bestelltes Forschungsfeld. Wu und Dredze fanden heraus, dass mBERT zwar nicht in allen 104 Sprachen der Trainingsdaten gleich gut abschneidet, aber bei Eigennamenerkennen, Part-of-Speech-Tagging und Dependency Parsing⁵⁰ bessere Ergebnisse vorweist als einsprachige Modelle, die mit vergleichbaren Datenmengen für vier Sprachen mit wenig Sprachmaterial trainiert wurden.⁵¹ Umgekehrt untersuchten Nozza u.a. einsprachige BERT-Modelle,⁵² die mit spezifischen Anpassungen ihrer Architektur oder zusätzlichen einsprachigen Daten entwickelt wurden, und stellten fest, dass sie mBERT bei 29 Aufgaben allgemein übertreffen. In jedem Fall haben diese Modelle zu den von Joshi u.a. aufgeworfenen Ausgrenzungsrisiken beigetragen: Über 90 % der weltweit von mehr als einer Milliarde Menschen gesprochenen Sprachen haben derzeit nur wenig oder gar keine sprachtechnologische Unterstützung.⁵³

⁴⁷ Devlin u.a., „BERT“.

⁴⁸ Linting Xue u.a., „mT5: A Massively Multilingual Pre-Trained Text-to-Text Transformer“, in: *arXiv*, Oktober 2020, DOI: 10.48550/arXiv.2010.11934.

⁴⁹ Brown u.a., „Language Models are Few-Shot Learners“.

⁵⁰ *Part of speech tagging* ist der automatisierte Prozess, den einzelnen Elementen eines Textes Wortarten (wie z.B. Substantiv, Verb, Adjektiv) zuzuweisen; *dependency parsing* meint eine Methode zur Analyse der grammatikalischen Struktur eines Satzes, wobei die Abhängigkeitsbeziehungen zwischen den Wörtern ermittelt und dargestellt werden. [Anm. d. Hg.]

⁵¹ Shijie Wu und Mark Dredze, „Are All Languages Created Equal in Multilingual BERT?“, in: *Proceedings of the 5th Workshop on Representation Learning for NLP* (2020), S. 120–130.

⁵² Debora Nozza, Federico Bianchi und Dirk Hovy, „What the [MASK]? Making Sense of Language-Specific BERT Models“, in: *arXiv*, März 2020, DOI: 10.48559/arXiv.2003.02912.

⁵³ Pratik Joshi u.a., „The State and Fate of Linguistic Diversity and Inclusion in the NLP World“, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), S. 6282–6293.

Neben Arbeiten, die untersuchen, welche Informationen die Modelle aus den Daten erhalten, sehen wir einen Trend zur Verringerung der Größe dieser Modelle durch verschiedene Techniken wie Wissensdestillation,⁵⁴ Quantisierung,⁵⁵ faktorierte Einbettungsparametrisierung und schichtenübergreifenden Parameterraustausch⁵⁶ sowie progressive Modulerersetzung.⁵⁷ Rogers u.a. bieten einen umfassenden Vergleich von Modellen,⁵⁸ die unter Verwendung dieser Techniken aus BERT abgeleitet wurden, etwa DistilBERT und ALBERT.⁵⁹ Während diese Modelle die Leistungsfähigkeit des ursprünglichen BERT-Modells trotz ihrer viel geringeren Größe beibehalten und manchmal sogar übertreffen, sind sie letztlich immer noch auf große Datenmengen und erhebliche Verarbeitungs- und Speicherkapazitäten angewiesen, um das Modell zu erstellen und zu reduzieren.

Wir stellen fest, dass der Wechsel von n-Gramm-Sprachmodellen zu Wortvektoren, die von neuronalen auf vortrainierte

⁵⁴ Cristian Bucilua, Rich Caruana und Alexandru Niculescu-Mizil, „Model Compression“, in: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2006), S. 535–541; Geoffrey Hinton, Oriol Vinyals und Jeff Dean, „Distilling the Knowledge in a Neural Network“, in: *arXiv*, März 2015, DOI: 10.488550/arXiv.1503.02531.

⁵⁵ Sheng Shen u.a., „Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT“, in: *arXiv*, September 2019, DOI: 10.48550/arXiv.1909.05840; Ofir Zafrir u.a., „Q8BERT: Quantized 8Bit BERT“, in: *arXiv*, Oktober 2019, DOI: 10.48550/arXiv.1910.06188.

⁵⁶ Lan u.a., „ALBERT“.

⁵⁷ Xu u.a., „BERT-of-Theseus“. Bei der *knowledge distillation* wird die Wissensrepräsentation eines großen, komplexen Sprachmodells in ein kleineres, effizienteres Modell übertragen; *quantization* ist eine Technik, bei der die Anzahl der Bits, die zur Darstellung der Gewichte eines neuronalen Netzes verwendet werden, reduziert wird; mit *factorized embedding parametrization* können große Embeddings in kleinere Teil-Embeddings zerlegt werden, um Speicherbedarf zu reduzieren; *cross-layer parameter sharing* teilt Gewichte zwischen verschiedenen Schichten eines neuronalen Netzes, um die Anzahl der zu trainierenden Parameter zu reduzieren; *progressive module replacing* ersetzt nach und nach Teile eines neuronalen Netzes durch effizientere Module während des Trainings. [Anm. d. Hg.]

⁵⁸ Anna Rogers, Olga Kovaleva und Anna Rumshisky, „A Primer in BERTology: What We Know About How BERT Works“, in: *Transactions of the Association for Computational Linguistics* 8 (2021), S. 842–866.

⁵⁹ Sanh u.a., „DistilBERT“; Lan u.a., „ALBERT“.

Transformer-Sprachmodelle destilliert wurden, mit einer Ausweitung und Veränderung der Aufgabentypen einhergeht, für die sie verwendet werden: n-Gramm-Sprachmodelle wurden typischerweise zunächst dafür eingesetzt, zwischen den Ausgaben von z.B. akustischen oder Übersetzungsmodellen eine Auswahl zu treffen. Die von LSTMs abgeleiteten Wortvektoren setzte man bald als effektivere Repräsentationen von Wörtern (statt der älteren *bag of words*-Merkmale)⁶⁰ in einer Vielzahl von NLP-Aufgaben ein, bei denen es um Labeling und Klassifizierung geht. Die vortrainierten Transformer-Modelle können darüber hinaus auf sehr kleine Datensätze umtrainiert werden (*few-shot*, *one-shot* oder sogar *zero-shot learning*), um scheinbar bedeutungsverarbeitende Aufgaben wie Zusammenfassung, Beantwortung von Fragen und Ähnliches auszuführen. Nichtsdestotrotz haben alle diese Systeme die Eigenschaft, Sprachmodelle im oben genannten Sinne zu sein – sie sind Systeme, die darauf trainiert wurden, Wörter (oder Zeichen oder Sätze) vorherzusagen. Sie unterscheiden sich in der Größe der Trainingsdatensätze, die sie nutzen, und in ihren potentiellen Anwendungsbereichen. Indem sie auf diese beiden Arten skaliert werden, ziehen moderne, extrem große Sprachmodelle neue Risiken auf sich, auf die wir in den folgenden Abschnitten eingehen.

3. Ökologische und finanzielle Kosten

Strubell u.a. haben kürzlich die Kosten – in Dollar wie in geschätzten CO₂-Emissionen – für die Entwicklung und das

⁶⁰ Ein *bag of words*-Modell ist eine Repräsentationsart, bei der ein Text als ungeordnete Sammlung von Wörtern dargestellt wird, wobei die Wortreihenfolge ignoriert und lediglich die Häufigkeit der einzelnen Wörter im Text berücksichtigt werden. [Anm. d. Hg.]

Training von Modellen verglichen.⁶¹ Während der durchschnittliche Mensch für schätzungsweise 5 Tonnen CO₂e⁶² pro Jahr verantwortlich ist,⁶³ trainierten die Autor:innen ein großes Transformer-Modell⁶⁴ mit neuronaler Architektursuche⁶⁵ und schätzten, dass das Trainingsverfahren 284 Tonnen CO₂ ausstieß. Das Training eines einzelnen BERT-Basismodells (ohne Hyperparameter-Abstimmung) auf GPUs⁶⁶ benötigte schätzungsweise so viel Energie wie ein Flug von der Ost- zur Westküste der USA.

Zwar kommt ein Teil dieser Energie aus erneuerbaren Quellen oder von Cloud-Computing-Unternehmen, welche die Emissionsschulden ausgleichen, doch stellen die Autor:innen fest, dass der Großteil dieser Energie nicht aus erneuerbaren Quellen stammt und viele Energiequellen nicht kohlenstoffneutral sind. Außerdem können auch erneuerbare Energiequellen noch auf verschiedene Weisen umweltschädlich sein⁶⁷ und Rechenzentren mit steigenden Rechenanforderungen anderen potenziellen Verwendungszwecken grüner Energie

⁶¹ Strubell, Ganesh und McCallum, „Energy“.

⁶² „CO₂e“ oder „Kohlendioxid-Äquivalent“ ist eine Einheit, mit der die Auswirkung verschiedener Treibhausgase in Bezug auf dieselbe Vergleichsmenge an CO₂ beschrieben wird. [Anm. d. Hg.]

⁶³ Daten für 2017, Hannah Ritchie, Max Roser und Pablo Rosado, „CO₂ and Greenhouse Gas Emissions“, in: *Our World in Data*, Mai 2020, <https://ourworldindata.org/co2-emissions> [Letzter Zugriff: 17.7.2023].

⁶⁴ Ashish Vaswani u.a., „Attention Is All You Need“, in: *Advances in Neural Information Processing Systems* (2017), S. 5999–6009.

⁶⁵ *Neural architecture search* ist ein Verfahren in der künstlichen Intelligenz zum automatischen Auffinden optimaler neuronaler Netzwerkarchitekturen für eine bestimmte Aufgabe oder einen bestimmten Datensatz, in dem verschiedene Strukturen und Hyperparameter getestet und bewertet werden. [Anm. d. Hg.]

⁶⁶ *Graphics processing unit*, die statt CPUs für das Training neuronaler Netze verwendet wird. [Anm. d. Hg.]

⁶⁷ David Bol, „14m Trees Have Been Cut down in Scotland to Make Way for Wind Farms“, in: *The Herald*, 29. Februar 2020, <https://www.heraldsotland.com/news/18270734.14m-trees-cut-scotland-make-way-wind-farms/> [Letzter Zugriff: 17.7.2023].

den Platz wegnehmen,⁶⁸ was den Bedarf an energieeffizienten Modellarchitekturen und Trainingsparadigmen unterstreicht.

Strubell u.a. untersuchten auch die Kosten dieser Modelle im Verhältnis zu dem erzielten Gewinn an Genauigkeit. Für die maschinelle Übersetzung, bei der große Sprachmodelle zu Leistungssteigerungen geführt haben, schätzen sie, dass eine Steigerung der BLEU-Punktzahl um 0,1 bei der Anwendung von neuronaler Architektursuche für die Übersetzung vom Englischen ins Deutsche (zusätzlich zu den Kohlendioxidemissionen) zu einem Anstieg der Rechenkosten um 150.000 US-Dollar führt. Um den Zugang zur NLP-Forschung niedrigschwelliger zu gestalten und den CO₂-Fußabdruck zu verringern, empfehlen die Autor:innen, die Trainingszeit und die Sensibilität der Hyperparameter anzugeben, wenn das einmal veröffentlichte Modell für eine spätere Verwendung neu trainiert werden soll. Außerdem fordern sie Regierungen dazu auf, in Compute Clouds zu investieren, um Forschenden einen gleichberechtigten Zugang zu ermöglichen. Initiativen wie der SustainNLP-Workshop⁶⁹ haben sich seitdem zum Ziel gesetzt, recheneffiziente Hardware und Algorithmen zu priorisieren. Schwartz u.a. fordern auch die Entwicklung einer grünen KI,⁷⁰ analog zu anderen, umweltfreundlichen wissenschaftlichen Entwicklungen wie der grünen Chemie oder dem nachhaltigen Computing. Wie Amodei und Hernandez gezeigt haben,⁷¹ hat sich der Rechenaufwand für das Training der größten Deep-Learning-Modelle (für NLP und andere Anwendungen) innerhalb von sechs Jahren um das 300.000-Fache erhöht und ist damit weit

⁶⁸ O.V., „Microsoft Announces One of the Largest Wind Deals in the Netherlands with Vattenfall“, 2. November 2017, <https://news.microsoft.com/2017/11/02/microsoft-announces-one-of-the-largest-wind-deals-in-the-netherlands-with-vattenfall/> [Letzter Zugriff: 17.7.2023].

⁶⁹ SustainNLP, „Organization“, <https://sites.google.com/view/sustainlp2020/organization> [Letzter Zugriff: 17.7.2023].

⁷⁰ Roy Schwartz u.a., „Green AI“, in: *Communications of the ACM* 63:12 (2020), S. 54–63.

⁷¹ Dario Amodei und Daniel Hernandez, „AI and Compute“, in: *OpenAI Blog*, 2018, <https://openai.com/blog/ai-and-compute/> [Letzter Zugriff: 17.7.2023].

schneller gewachsen als das Moore'sche Gesetz. Um grüne KI zu fördern, plädieren Schwartz u.a. dafür, Effizienz in den Bewertungsmaßstab aufzunehmen. Sie zeigen, dass die meisten Beiträge der führenden Konferenzen – ACL 2018, NeurIPS 2018 und CVPR 2019 – allein Verbesserungen in der Genauigkeit als ihren primären Beitrag zum Fachgebiet anführen und sich keiner von ihnen auf Maßnahmen zur Effizienzsteigerung konzentriert. Seitdem sind verschiedene Online-Tools veröffentlicht worden,⁷² die Forschende beim Benchmarking ihres Energieverbrauchs helfen. Unter anderem empfehlen diese, Experimente in Regionen mit geringem CO₂-Ausstoß durchzuführen, konsequent Energie- und CO₂-Kennzahlen anzugeben und Kompromisse bei der Energieleistung in Betracht zu ziehen, bevor besonders energiehungrige Modelle eingesetzt werden. Zusätzlich zu diesen Aufrufen, die Forschung und Entwicklung besser zu dokumentieren und technische Korrekturen vorzunehmen, betonen Bietti und Vatanparast die Notwendigkeit von sozialem und politischem Engagement zugunsten der Gestaltung einer Zukunft, in der datengesteuerte Systeme nur noch minimale negative Auswirkungen auf die Umwelt haben.⁷³

Während Strubell u.a. bereits ein Benchmarking des Trainingsprozesses in einer spezifischen Forschungsumgebung durchgeführt haben,⁷⁴ werden viele Sprachmodelle so eingesetzt – sei es in der Industrie oder in anderen Bereichen –, dass die Kosten für die Anwendung [*inference cost*] die Kosten für das Training auf lange Sicht bei Weitem übersteigen können. In diesem Szenario kann es sinnvoller sein, Modelle mit in der Anwendungsphase geringeren Energiekosten einzusetzen,

⁷² Henderson u.a., „Systematic Reporting“; Kadan Lottick u.a., „Energy Usage Reports: Environmental awareness as part of algorithmic accountability“, in: *arXiv*, Dezember 2019, DOI: 10.48550/arXiv.1911.08354.

⁷³ Elettra Bietti und Roxana Vatanparast, „Data Waste“, in: *Harvard International Law Journal* 61 (2020), S. I–II.

⁷⁴ Strubell, Ganesh und McCallum, „Energy“.

auch wenn ihre Trainingskosten hoch sind. Zusätzlich zu den Benchmarking-Tools wird Forschung zur Abschätzung des Kostenanstiegs, der mit der Einführung von Sprachmodellen für bestimmte Anwendungen verbunden ist, sowie deren Abschneiden gegenüber alternativen NLP-Methoden immer wichtiger werden, um das je notwendige Quidproquo zu verstehen.

Bei der Durchführung von Risiko-Nutzen-Analysen im Bereich von Sprachtechnologien ist es wichtig zu berücksichtigen, wie Risiken und Nutzen genau verteilt sind, da sie nicht gleichmäßig dieselben Personen betreffen. In der Literatur über Umweltrassismus wurde hinreichend nachgewiesen, dass die negativen Auswirkungen des Klimawandels die weltweit am stärksten marginalisierten Gruppen zuerst erreichen und beeinträchtigen.⁷⁵ Ist es fair oder gerecht zu verlangen, dass zum Beispiel die Bewohner:innen der Malediven (die wahrscheinlich bis zum Jahr 2100 überflutet sein werden)⁷⁶ oder die 800.000 Menschen im Sudan, die unter drastischen Überschwemmungen zu leiden haben,⁷⁷ den ökologischen Preis für die Entwicklung und den Einsatz immer größerer englischer Sprachmodelle zahlen, während ähnliche groß angelegte Modelle nicht für die maledivische Sprache Dhivehi oder das sudanesisch Arabisch produziert werden?⁷⁸

⁷⁵ Hussein M. Adam, Robert D. Bullard und Elizabeth Bell, *Faces of Environmental Racism: Confronting Issues of Global Justice*, Totowa: Rowman & Littlefield 2001; Robert D. Bullard, *Confronting Environmental Racism: Voices from the Grassroots*, Boston: South End Press 1993; O.V., „Report: Inequalities Exacerbate Climate Impacts on Poor“, in: *United Nations Sustainable Development Blog*, Oktober 2016, <https://www.un.org/sustainabledevelopment/blog/2016/10/report-inequalities-exacerbate-climate-impacts-on-poor/> [Letzter Zugriff: 17.7.2023].

⁷⁶ David Anthoff, Robert J Nicholls und Richard SJ Tol, „The Economic Impact of Substantial Sea-Level Rise“, in: *Mitigation and Adaptation Strategies for Global Change* 15:4 (2010), S. 321–335.

⁷⁷ O.V., „Floods-Hit Sudan Facing ‘Unprecedented Challenges’, UN Warns“, 25. September 2020, <https://www.aljazeera.com/news/2020/9/25/over-800000-affected-in-sudan-flooding-un> [Letzter Zugriff: 17.7.2023].

⁷⁸ Mit dieser Bemerkung wollen wir die existierenden Arbeiten zu ressourcenarmen Sprachen nicht übergehen. Ein besonders spannendes Beispiel ist das Masakhane-Projekt, das partizipative Forschungstechniken für die Entwicklung von maschineller

Und während einige Sprachtechnologien wirklich dazu gedacht sind, marginalisierten Gruppen zu helfen,⁷⁹ werden die meisten von ihnen entwickelt, um die Bedürfnisse derjenigen zu befriedigen, die bereits über die größten gesellschaftlichen Privilegien verfügen. Man bedenke etwa, wer sowohl die finanziellen Mittel hat, um ein Google Home, Amazon Alexa oder ein Apple-Gerät mit installiertem Siri zu kaufen, als auch eine Sprache spricht, die diese Devices problemlos verstehen. Wenn große Sprachmodelle hegemoniale Vorurteile codieren und verstärken (siehe dazu die Abschnitte 4 und 6), werden die daraus resultierenden Schäden höchstwahrscheinlich marginalisierte Bevölkerungsgruppen treffen, die selbst in reichen Nationen am ehesten von Umweltrassismus betroffen sind.⁸⁰

Diese Modelle werden in einer Zeit entwickelt, in der weltweit noch nie dagewesene Umweltveränderungen zu beobachten sind. Vom Monsunregen, der durch neue Niederschlagsmuster aufgrund des Klimawandels verursacht wird und von dem mehr als 8 Millionen Menschen in Indien betroffen sind,⁸¹ bis hin zur schlimmsten je gemessenen Feuersaison in Australien, bei der fast drei Milliarden Tiere und mindestens 400 Men-

Übersetzung für afrikanische Sprachen untersucht: Wilhelmina Nekoto u.a., „Participatory Research for Low-Resourced Machine Translation: A Case Study in African Languages“, in: *Findings of the Association for Computational Linguistics: EMNLP 2020* (2020), S. 2144–2160. Diese vielversprechenden Ansätze erfordern nicht die Anhäufung von Terabytes an Daten.

⁷⁹ Steven Bird, „Social Mobile Technologies for Reconnecting Indigenous and Immigrant Communities“, in: *People.Policy.Place Seminar. Northern Institute, Charles Darwin University* (2016); Aidan Pine und Mark Turin, „Language Revitalization“, in: *Oxford Research Encyclopedia of Linguistics* (2017).

⁸⁰ Russel Barsh, „Indigenous Peoples, Racism and the Environment“, in: *Meanjin* 49:4 (1990), S. 723; Laura Pulido, „Flint, Environmental Racism, and Racial Capitalism“, in: *Capitalism Nature Socialism* 27:3 (2016), S. 1–16.

⁸¹ Anjana Pasricha, „Monsoons Cause Havoc in India as Climate Change Alters Rainfall Patterns“, 14. August 2020, https://www.voanews.com/a/south-central-asia_monsoons-cause-havoc-india-climate-change-alters-rainfall-patterns/6194416.html [Letzter Zugriff: 17.7.2023].

schen getötet oder vertrieben wurden,⁸² stellen die Auswirkungen der Klimaveränderung jedes Jahr neue Rekorde auf. Es ist an der Zeit, dass Forschende den Fragen von Energieeffizienz und möglichen Schäden Vorrang einräumen, um die negativen Auswirkungen auf die Umwelt und den ungleichen Zugang zu Ressourcen zu verringern – denn beides trifft Menschen, die ohnehin schon am gesellschaftlichen Rand existieren, unverhältnismäßig stark.

4. Unergründliche Trainingsdaten

Die schiere Masse der im Internet verfügbaren Daten hat es Deep-Learning-Modellen ermöglicht, bei bestimmten Benchmarks in NLP- und Computer-Vision-Anwendungen eine hohe Genauigkeit zu erreichen. In beiden Anwendungsbereichen hat sich jedoch gezeigt, dass die Trainingsdaten problematische Eigenschaften aufweisen,⁸³ was zu Modellen führt, die stereotype und abwertende Assoziationen entlang der Achsen *gender*, *race*, ethnischer Zugehörigkeit und Behinderungsstatus codieren.⁸⁴ In diesem Abschnitt erörtern wir, wie große, unkuratierte,

⁸² O.V., „Australia Fires: Nearly Three Billion Animals Killed or Displaced“, 28. Juli 2020, <https://edition.cnn.com/2020/07/28/asia/australia-fires-wildlife-report-scli-intl-scn/index.html> [Letzter Zugriff: 17.7.2023].

⁸³ Abeba Birhane und Vinay Uday Prabhu, „Large Image Datasets: A Pyrrhic Win for Computer Vision?“, in: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (2021), S. 1537–1547; Terrance de Vries u.a., „Does Object Recognition Work for Everyone?“, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2019), S. 52–59; Ethan Fast, Tina Vachovsky und Michael Bernstein, „Shirtless and Dangerous: Quantifying Linguistic Signals of Gender Bias in an Online Fiction Writing Community“, in: *Proceedings of the International AAAI Conference on Web and Social Media* 10 (2016); Antigoni Founta u.a., „Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior“, in: *Proceedings of the International AAAI Conference on Web and Social Media* 12 (2018); Ben Hutchinson u.a., „Social Biases in NLP Models as Barriers for Persons with Disabilities“, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), S. 5491–5501.

⁸⁴ Christine Basta, Marta R. Costa-Jussà und Noe Casas, „Evaluating the Underlying Gender Bias in Contextualized Word Embeddings“, in: *Proceedings of the First Workshop on Gender Bias in Natural Language Processing* (2019), S. 33–39; Iz Beltagy, Kyle Lo und

internetbasierte Datensätze die dominanten und hegemonialen Sichtweisen codieren, die Marginalisierten weiter schaden; wir empfehlen, erhebliche Ressourcen für die Kuratierung von Datensätzen und Dokumentationsverfahren bereitzustellen.

4.1 Größe ist kein Garant für Vielfalt

Das Internet ist ein großer und vielfältiger virtueller Raum, und man erkennt leicht, dass sehr große Datensätze wie Common Crawl („Petabytes an Daten, die über ein acht Jahre dauerndes Web-Crawling gesammelt wurden“, ⁸⁵ von denen eine gefilterte Version in den GPT-3-Trainingsdaten enthalten ist) daher weitgehend repräsentativ für die Art und Weise sind, wie Menschen die Welt sehen. Bei näherer Betrachtung ist aber festzustellen, dass mehrere Faktoren die Internetbeteiligung wie die Diskussionen, die durch die Crawling-Methode erfasst werden, einschränken und schließlich auch die Texte, die nach dem Filtern der gecrawlten Daten enthalten sind, wenig repräsentativ sind. In allen Fällen sind es mehrheitlich die Stimmen von Personen, die am ehesten einen hegemonialen Standpunkt vertreten, die auch im Trainingsdatensatz erhalten bleiben. Im Falle des US-amerikanischen und britischen Englisch bedeutet dies, dass weiß-rechtsextreme, frauenfeindliche, altersdiskriminierende usw. Ansichten in den Trainingsdaten überrepräsentiert sind;

Arman Cohan, „SciBERT: A Pretrained Language Model for Scientific Text“, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (2019), S. 3615–3620; Keita Kurita u.a., „Measuring Bias in Contextualized Word Representations“, in: *Proceedings of the First Workshop on Gender Bias in Natural Language Processing* (2019), S. 166–172; Yi Chern Tan und L Elisa Celis, „Assessing Social and Intersectional Biases in Contextualized Word Representations“, in: *Advances in Neural Information Processing Systems* (2019), S. 13230–13241; Jieyu Zhao u.a., „Gender Bias in Contextualized Word Embeddings“, in: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies 1* (2019), S. 629–634.

⁸⁵ „Common Crawl“, <https://commoncrawl.org/> [Letzter Zugriff: 17.7.2023].

sie übersteigen nicht nur die relative Häufigkeit in der Allgemeinbevölkerung – diese Tatsache führt auch dazu, dass Modelle, die mit diesen Datensätzen trainiert wurden, existierende Biases⁸⁶ und mögliche Schäden weiter verstärken.

Fragt man, wer zu diesen Internet-Textsammlungen beiträgt, sehen wir, dass der Zugang zum Internet selbst nicht gleichmäßig verteilt ist, was dazu führt, dass in den Internetdaten jüngere Nutzer:innen und solche aus Industrieländern überrepräsentiert sind.⁸⁷ Es geht nicht um das Internet als Ganzes, sondern vielmehr um bestimmte Teilbereiche davon. Die Trainingsdaten von GPT-2 stammen beispielsweise aus dem *scraping*⁸⁸ ausgehender Links der Plattform Reddit; eine Pew-Umfrage von 2016 zeigte, dass 67 % der Reddit-Nutzer:innen in den Vereinigten Staaten Männer sind, und 64 % sind zwischen 18 und 29 Jahre alt.⁸⁹ Ganz ähnlich zeigen aktuelle Untersuchungen zu Wikipedia-Beitragenden, dass nur 8,8–15 % von ihnen Frauen oder Mädchen sind.⁹⁰

Darüber hinaus stellen sich nutzer:innengenerierte Inhaltsseiten wie Reddit, Twitter und Wikipedia zwar als offen und für jedermann zugänglich dar, doch gibt es strukturelle Fakto-

⁸⁶ Mit *bias* sind systematische Verzerrungen, bewusste Vorurteile oder unbewusste Bewertungstendenzen wie Rassismus oder Sexismus gemeint, deren latentes oder explizites Vorhandensein im Trainingsset vom Modell gelernt und reproduziert werden kann. Der Begriff stammt ursprünglich aus der Kognitionswissenschaft, hat sich aber gerade bei der Diskussion der politischen Konsequenzen von KI etabliert, weshalb er hier als Fremdwort beibehalten wird. [Anm. d. Hg.]

⁸⁷ Pine und Turin, „Language Revitalization“; World Bank, „Individuals Using the Internet“, <https://data.worldbank.org/indicator/IT.NET.USER.ZS?end=2017&location=s=US&start=2015> [Letzter Zugriff: 17.7.2023]. Diese Tatsache findet auch in der Modellkarte für GPT-3 Erwähnung: <https://github.com/openai/gpt-3/blob/master/model-card.md> [Letzter Zugriff: 17.7.2023].

⁸⁸ *Scraping* bezeichnet das automatische Extrahieren von Daten aus Webseiten. [Anm. d. Hg.]

⁸⁹ Michael Barthel u.a., „Seven-In-Ten Reddit Users Get News on The Site“, <https://www.journalism.org/2016/02/25/reddit-news-users-more-likely-to-be-male-young-and-digital-in-their-news-preferences/> [Letzter Zugriff: 17.7.2023].

⁹⁰ Amanda Lazar u.a., „Going Gray, Failure to Hire, and the Ick Factor: Analyzing How Older Bloggers Talk About Ageism“, in: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing* (2017), S. 655–668.

ren, einschließlich Moderationspraktiken, die sie für marginalisierte Bevölkerungsgruppen weniger einladend machen. Leslie Kay Jones⁹¹ dokumentiert (unter Verwendung von Techniken der digitalen Ethnografie)⁹² mehrere Fälle, in denen die Accounts von Personen, die auf Twitter Todesdrohungen erhalten haben, gesperrt wurden, während die Konten der Personen, die diese Drohungen aussprachen, weiter existierten. Sie berichtet überdies, dass Belästigungen auf Twitter von „einer Vielzahl von sich überschneidenden Gruppen erlebt werden, darunter Opfer von häuslichem Missbrauch, Sex Workers, Trans-Personen, queere Menschen, Immigrant:innen, medizinische Patient:innen (durch ihre Betreuer:innen), neurodiverse Menschen und Menschen mit Seh- oder Hörbeeinträchtigungen.“ Das Ergebnis ist, dass eine kleine Zahl weiterhin problemlos Daten hinzufügen, ihre Gedanken teilen und Plattformen entwickeln kann, die ihren Weltanschauungen entsprechen; dieses systemimmanente Muster verschlechtert aber im Gegenzug die Diversität und Inklusion von Kommunikation im Netz und schafft eine Feedback-Schleife, die den Einfluss von Daten aus unterrepräsentierten Bevölkerungsgruppen verringert.

Selbst wenn Gruppen, die sich auf den Mainstream-Websites nicht willkommen fühlen, andere Kommunikationsforen einrichten, werden sie mit nur geringer Wahrscheinlichkeit als Trainingsdaten in Sprachmodelle Eingang finden. Nehmen wir zum Beispiel ältere Erwachsene in den USA und in Großbritannien. Lazar u.a. beschreiben, wie diese individuell und kollektiv ihre Haltung gegen Altersdiskriminierung artikulieren, insbeson-

⁹¹ Leslie Kay Jones, „Twitter Wants You to Know That You're Still SOL If You Get a Death Threat – Unless You're President Donald Trump“, in: *Medium*, Oktober 2020, <https://medium.com/@agua.carbonica/twitter-wants-you-to-know-that-youre-still-sol-if-you-get-a-death-threat-unless-you-re-a5c316b706> [Letzter Zugriff: 17.7.2023].

⁹² Leslie Kay Jones, „#BlackLivesMatter: An Analysis of the Movement as Social Drama“, in: *Humanity & Society* 44:1 (2020), S. 92–110.

dere über Blogs,⁹³ die einige ältere Erwachsene den populäreren Social-Media-Seiten vorziehen, um sensible Themen zu diskutieren.⁹⁴ In diesen Foren wird ausgiebig darüber diskutiert, was Altersdiskriminierung ist und welche Auswirkungen sie hat. Eine Blogging-Community wie die von Lazar u.a. beschriebene, ist jedoch weniger einfach zu finden als andere Blogs, die mehr eingehende und ausgehende Links haben.

Schließlich kann die derzeitige Praxis des Filterns von Datensätzen die Stimmen von Menschen mit marginalisierten Identitäten weiter zum Schweigen bringen. Der Trainingsdatensatz für GPT-3 besteht aus einer gefilterten Version des Common Crawl-Datensatzes, die durch das Training eines Klassifikators entwickelt wurde, um die Dokumente herauszufiltern, die den in den Trainingsdaten von GPT-2 verwendeten am ähnlichsten sind, also Dokumente, die von Reddit, Wikipedia und einer Sammlung von Büchern aus verlinkt wurden.⁹⁵ Berichten zufolge hatte man auf diese Weise zwar Dokumente herausgefiltert, die frühere Arbeiten als „unverständlich“ bezeichnet hatten,⁹⁶ aber nicht gemessen wurde (und daher unbekannt ist), was sonst noch der Datensatzsäuberung zum Opfer fiel. Das Colossal Clean Crawled Corpus,⁹⁷ das in Fedus u.a. zum Trainieren eines Sprachmodells mit Billionen von Parametern verwendet wurde,⁹⁸ wird unter anderem dadurch bereinigt, dass jede Seite aussortiert wird, die eines von etwa 400 „schmutzigen, unanständigen, obszönen oder anderweitig

⁹³ Lazar u.a., „Going Gray, Failure to Hire, and the Ick Factor: Analyzing How Older Bloggers Talk About Ageism“.

⁹⁴ Robin Brewer und Anne Marie Piper, „Tell It Like It Really Is: A Case of Online Content Creation and Sharing Among Older Adult Bloggers“, in: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (2016), S. 5529–5542.

⁹⁵ Brown u.a., „Language Models are Few-Shot Learners“.

⁹⁶ Trieu H Trinh und Quoc V Le, „A Simple Method for Commonsense Reasoning“, in: *arXiv*, Juni 2018, DOI: 10.48550/arXiv.1806.02847.

⁹⁷ Raffel u.a., „Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer“.

⁹⁸ Fedus, Zoph und Shazeer, „Switch Transformers“.

‚bösen‘ Wörtern“ enthält.⁹⁹ Diese Liste besteht überwiegend aus Wörtern, die mit Sex zu tun haben, sowie aus einer Handvoll rassistischer Schimpfwörter und Vokabeln der *white supremacy* (z.B. ‚Hakenkreuz‘, ‚White Power‘). Dieser Ansatz ist zwar möglicherweise wirksam bei der Entfernung von Dokumenten, die Pornografie (und die damit verbundenen problematischen Stereotypen, die in der Sprache solcher Seiten codiert sind)¹⁰⁰ und bestimmte Arten von Hassreden enthalten, schwächt aber zweifellos auch den Einfluss von Online-Räumen, die von und für LGBTQ-Personen geschaffen wurden, indem etwa Wörter wie *twink* unterdrückt werden.¹⁰¹ Wenn wir den Diskurs von marginalisierten Bevölkerungsgruppen herausfiltern, scheitern wir daran, Trainingsdaten bereitzustellen, die einst negativ konnotierte Wörter neu und positiv besetzen und marginalisierte Identitäten selbstbestimmt in einem positiven Licht beschreiben können.

Bei jedem Schritt – von der anfänglichen Teilnahme an Internetforen über die fortgesetzte Präsenz dort bis hin zur Sammlung und schließlich der Filterung von Trainingsdaten – wird also in der derzeitigen Praxis der hegemoniale Standpunkt priorisiert. Indem wir große Mengen von Webtexten als ‚repräsentativ‘ für ‚die gesamte Menschheit‘ betrachten, riskieren wir, dominante Sichtweisen zu fixieren, Machtungleichgewichte zu vergrößern und Ungleichheit weiter zu festigen. Wir schlagen stattdessen Praktiken vor, die aktiv versuchen, Gemeinschaften einzubeziehen, die im Internet unterrepräsentiert sind. So sind Dekolonisierungsbestrebungen im Bildungswesen beispielhaft, bei denen man sich aufgrund der Überrepräsentation kolonialer Ansichten in Texten hin zu

⁹⁹ Verfügbar unter <https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/blob/master/en> [Letzter Zugriff: 17.7.2023].

¹⁰⁰ Robyn Speer, „ConceptNet Numberbatch 17.04: Better, Less-Stereotyped Word Vectors“, <https://blog.conceptnet.io/2017/04/24/conceptnet-numberbatch-17-04-better-less-stereotyped-word-vectors/> [Letzter Zugriff: 17.7.2023].

¹⁰¹ Diese Beobachtung verdanken wir William Agnew.

mündlichen Überlieferungen bewegt.¹⁰² Überdies können Trainingsdatensätze in einem behutsamen Prozess kuratiert werden, in dessen Folge man entscheidet, was in sie hineingehört, anstatt nur auf die Größe zu achten und wahllos zu versuchen, Treibgut, das als ‚gefährlich‘, ‚unverständlich‘ oder ‚anderweitig schlecht‘ eingestuft wurde, nachträglich auszusortieren.

4.2 Statische Daten vs. wechselnde soziale Ansichten

Ein zentraler Aspekt bei der Entstehung sozialer Bewegungen ist der strategische Einsatz von Sprache, um dominante Narrative zu destabilisieren und unterrepräsentierten gesellschaftlichen Perspektiven Aufmerksamkeit zu verschaffen. Soziale Bewegungen bringen neue Normen, Sprach- und Kommunikationsformen hervor. Das ist eine zusätzliche Herausforderung für den Einsatz von Sprachmodellen, weil Methoden, die sich auf diese Modelle stützen, das Risiko einer ‚Wertesperre‘ [value lock] bergen, bei der die Technologie ältere, weniger inklusive Auffassungen auf Dauer stellt.

Mit dem Erfolg der Black-Lives-Matter-Bewegung (BLM) vermehrte sich etwa die Erstellung und Bearbeitung von Wikipedia-Artikeln über Erschießungen von Schwarzen Menschen.¹⁰³ Es zeigte sich, dass Artikel, die frühere Erschießungen und Vorfälle von Polizeibrutalität beschreiben, im gleichen Zug

¹⁰² Benjamin Dangl, *The Five Hundred Year Rebellion: Indigenous Movements and the Decolonization of History in Bolivia*, Chico, CA: AK Press 2019; Mette Edith Lundsfryd, „Speaking Back to a World of Checkpoints: Oral History as a Decolonizing Tool in the Study of Palestinian Refugees from Syria in Lebanon“, in: *Middle East Journal of Refugee Studies* 2:1 (2017), S. 73–95; Katrina Srigley und Lorraine Sutherland, „Decolonizing, Indigenizing, and Learning Biskaaybiyang in the Field: Our Oral History Journey“, in: *The Oral History Review* 45:1 (2019), S. 7–28.

¹⁰³ Marlon Twyman, Brian C. Keegan und Aaron Shaw, „Black Lives Matter in Wikipedia: Collective Memory and Collaboration Around Online Social Movements“, in: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing* (2017), S. 1400–1412.

erstellt und aktualisiert wurden, wie Artikel für neuere Ereignisse hinzukamen. Soziale Bewegungen stellen so Verbindungen zwischen historischen und gegenwärtigen Ereignissen her, um kohärente Erzählungen zu formen.¹⁰⁴ Twyman u.a. betonen, wie soziale Bewegungen aktiv auf Framings und Reframings von Minderheitsnarrativen in gerade solchen Online-Diskursen Einfluss nehmen, die potenziell jene Daten bilden, die Sprachmodellen zugrunde liegen.¹⁰⁵

Ein wichtiger zu erwähnender Vorbehalt ist, dass schlecht dokumentierte soziale Bewegungen ohne große Medienaufmerksamkeit überhaupt nicht erfasst werden. Die Berichterstattung kann dann Proteste und soziale Bewegungen vernachlässigen¹⁰⁶ oder Ereignisse, die die Staatsmacht herausfordern, direkt verzerren.¹⁰⁷ Beispiele sind Medienhäuser, die friedliche Proteste ignorieren und sich stattdessen auf dramatische oder gewalttätige Ereignisse konzentrieren, die zwar guten Einschaltquoten nützen, aber fast immer in einseitiger Berichterstattung resultieren.¹⁰⁸ Infolgedessen können die Daten, die den Sprachmodellen zugrunde liegen, soziale Bewegungen verzerrt darstellen und sich unverhältnismäßig stark an bestehende Machtssysteme anlehnen.

Die Entwicklung und Verschiebung solcher Framings mag in großen Sprachmodellen auf unvollständige Weise erlernt werden oder in den Datensätzen untergehen – insbesondere, wenn diese nicht kontinuierlich aktualisiert werden. In Anbetracht der Kosten, die allein für das Training großer Sprachmo-

¹⁰⁴ Francesca Polletta, „Contending Stories: Narrative in Social Movements“, in: *Qualitative Sociology* 21:4 (1998), S. 419–446.

¹⁰⁵ Twyman, Keegan und Shaw, „BLM in Wikipedia“.

¹⁰⁶ Jennifer Earl u.a., „The Use of Newspaper Data in the Study of Collective Action“, in: *Annual Review of Sociology* 30 (2004), S. 65–80; David Ortiz u.a., „Where do we stand with newspaper data?“, in: *Mobilization: An International Quarterly* 10:3 (2005), S. 397–419.

¹⁰⁷ Christian Davenport, *Media Bias, Perspective, and State Repression: The Black Panther Party*, Cambridge, MA: Cambridge University Press 2009.

¹⁰⁸ Douglas M. McLeod, „News Coverage and Social Protest: How the Media’s Protect Paradigm Exacerbates Social Conflict“, in: *Journal of Dispute Resolution* 12:1 (2007), S. 1–10.

delle anfallen, ist es selbst für große Unternehmen wahrscheinlich nicht machbar, sie häufig genug vollständig neu aufzusetzen, um mit der hier diskutierten Art des Sprachwandels Schritt zu halten. Womöglich könnte hier gezieltes Finetuning¹⁰⁹ Abhilfe schaffen, um Sprachmodelle neu zu trainieren – aber auch hier sind durchdachte Kuratierungspraktiken zu wünschen, um geeignete Daten zur Erfassung solcher Reframings ausfindig zu machen. Es braucht überdies Bewertungsmaßstäbe dafür, ob Finetunings die Art und Weise, in der neue Rahmungen hegemoniale Repräsentationen angreifen, tatsächlich wirken.

4.3 Codierter Bias

Es ist mittlerweile breit anerkannt, dass große Sprachmodelle verschiedene Arten von Bias aufweisen, einschließlich stereotyper Assoziationen¹¹⁰ oder negativer Gefühle gegenüber bestimmten Gruppen.¹¹¹ Darüber hinaus sehen wir die Auswirkungen von Intersektionalität,¹¹² bei der BERT, ELMo, GPT und GPT-2 mehr Bias gegenüber intersektional marginalisierten Identitäten codieren, als man allein aufgrund der Kombination

¹⁰⁹ *Finetuning* bezeichnet das nach Abschluss des Grundtrainings erfolgende Nachtraining der letzten Schichten eines Sprachmodells auf einem vergleichsweise kleinen Datensatz, um Änderungen im ‚Stil‘ seiner Ausgaben hervorzurufen; Finetuning ist sehr viel weniger rechenintensiv als das Grundtraining. [Anm. d. Hg.]

¹¹⁰ Basta, Costa-Jussà und Casas, „Evaluating“; Beltagy, Lo und Cohan, „SciBERT“; Kurita u.a., „Measuring Bias in Contextualized Word Representations“; Emily Sheng u.a., „The Woman Worked as a Babysitter: On Biases in Language Generation“, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (2019), S. 3407–3412; Haoran Zhang u.a., „Hurtful Words: Quantifying Biases in Clinical Contextual Word Embeddings“, in: *Proceedings of the ACM Conference on Health, Inference, and Learning* (2020), S. 110–120; Zhao u.a., „Gender Bias“.

¹¹¹ Hutchinson u.a., „Social Biases in NLP Models“.

¹¹² Kimberlé Crenshaw, „Demarginalizing the Intersection of Race and Sex: A Black Feminist Critique of Antidiscrimination Doctrine, Feminist Theory and Antiracist Politics“, in: *The University of Chicago Legal Forum* 1989 (1989), S. 139.

jeder der Achsen erwarten würde.¹¹³ Viele dieser Arbeiten kommen zu dem Schluss, dass derartige Probleme auf die Eigenschaften der Trainingsdaten zurückzuführen sind. So stellen Hutchinson u.a. fest, dass BERT Phrasen, die sich auf Menschen mit Behinderungen beziehen, mit negativeren Stimmungswörtern assoziiert und dass Waffengewalt, Obdachlosigkeit und Drogenabhängigkeit in Texten, die psychische Erkrankungen behandeln, überrepräsentiert sind.¹¹⁴ In ähnlicher Weise zeigen Gehman u.a., dass Modelle wie GPT-3, die mit mindestens 570 GB, größtenteils aus Common Crawl¹¹⁵ stammenden Daten trainiert wurden, Sätze mit hohen Toxizitätswerten generieren können, selbst wenn sie mit nicht toxischen Sätzen gepromptet werden.¹¹⁶ Bei der Untersuchung der Trainingsdaten von GPT-2¹¹⁷ wurden außerdem 272.000 Dokumente von unzuverlässigen Nachrichtenseiten und 63.000 aus gesperrten Subreddits gefunden.

Diese Demonstrationen von Verzerrungen, die von Sprachmodellen erlernt werden, sind äußerst wertvoll, um sich des Schadenspotenzials bewusst zu werden, welches entsteht, wenn solche Modelle entweder bei der Texterzeugung oder als Komponenten von Klassifikationssystemen eingesetzt werden, wie wir in Abschnitt 6 näher erläutern. Sie stellen jedoch aus mehreren Gründen keine Methode dar, mit der sich alle derartigen Risiken erschöpfend aufdecken lassen.

¹¹³ Wei Guo und Aylin Caliskan, „Detecting Emergent Intersectional Biases: Contextualized Word Embeddings Contain a Distribution of Human-Like Biases“, in: *arXiv*, Juni 2020, DOI: 10.48550/arXiv.2006.03955.

¹¹⁴ Hutchinson u.a., „Social Biases in NLP Models“.

¹¹⁵ „Data“, <https://commoncrawl.org/the-data/> [Letzter Zugriff: 17.7.2023].

¹¹⁶ Samuel Gehman u.a., „RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models“, in: *Findings of the Association for Computational Linguistics: EMNLP 2020* (2020), S. 3356–3369.

¹¹⁷ Die Trainingsdaten von GPT-3 sind nicht öffentlich zugänglich, aber die Trainingsdaten von GPT-2 wurden indirekt verwendet, um die von GPT-3 zu konstruieren, siehe Gehman u.a.

Erstens stützen sich Modellprüfungsverfahren in der Regel auf automatisierte Systeme zur Sentiment- und Toxizitätsanalyse oder neuartiger Metriken wie *regard* (Beachtung) zur Messung der Einstellung gegenüber einer bestimmten demografischen Gruppe.¹¹⁸ Diese Systeme selbst sind jedoch möglicherweise keine zuverlässigen Mittel zur Messung der Toxizität solcher von Sprachmodellen erzeugten Texte. So hat sich beispielsweise herausgestellt, dass das Perspective-API-Modell Sätze, die Identitätsmarker für Randgruppen oder sogar bestimmte Namen enthalten, mit einem höheren Grad an Toxizität in Verbindung bringt.¹¹⁹

Zweitens erfordert ein Sprachmodell auf Biases zu überprüfen, *a priori* ein Verständnis der jeweils relevanten sozialen Kategorien. Die oben zitierten Arbeiten gehen im Allgemeinen von speziell US-amerikanischen Merkmalen wie *race* und *gender* aus (so, wie sie in den USA verstanden werden). Aber natürlich sind solche grundrechtlich geschützten Attribute [*protected attributes*] nicht die einzigen Identitätsmerkmale, die Gegenstand von Vorurteilen oder Diskriminierung sein können; wie die hervorstechenden Identitätsmerkmale sind auch Vorurteile kulturabhängig.¹²⁰ Daher benötigen Komponenten wie Toxizitätsklassifikatoren für jeden Prüfungskontext kulturrell angemessene Trainingsdaten – und selbst dann können wir marginalisierte Identitäten übersehen, wenn wir nicht wissen, worauf wir achten müssen.

¹¹⁸ Sheng u.a., „Babysitter“.

¹¹⁹ Hutchinson u.a., „Social Biases in NLP Models“; Vinodkumar Prabhakaran, Ben Hutchinson und Margaret Mitchell, „Perturbation Sensitivity Analysis to Detect Unintended Model Biases“, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing* (2019), S. 5740–5745.

¹²⁰ Susan T. Fiske, „Prejudices in Cultural Contexts: Shared Stereotypes (Gender, Age) Versus Variable Stereotypes (Race, Ethnicity, Religion)“, in: *Perspectives on Psychological Science* 12:5 (2017), S. 791–799; Sabine Sczesny u.a., „Gender Stereotypes and the Attribution of Leadership Traits: A Cross-Cultural Comparison“, in: *Sex Roles* 51:11–12 (2004), S. 631–645.

Schließlich stellen wir fest, dass der Schritt vom Nachweis von Vorurteilen zum Aufbau von Mechanismen, mit denen man die ‚Sicherheit‘ von Sprachmodellen (selbst für eine bestimmte grundrechtlich geschützte Gruppe) überprüft, eine Auseinandersetzung mit den Systemen der Macht erfordert, die zu eben jenen negativen Ergebnissen führen kann, die ein solcher Mechanismus zu verhindern sucht.¹²¹ So hat beispielsweise die #MeToo-Bewegung weitreichende Debatten über unangemessenes sexuelles Verhalten von Männern speziell in Machtpositionen, aber auch ganz im Allgemeinen angestoßen.¹²² Diese Diskussionen stellen Verhaltensweisen infrage, die in der Vergangenheit als angemessen oder sogar als die Schuld von Frauen betrachtet wurden, und verschieben die Vorstellung davon, was sexuell unangemessenes Verhalten ist. Jedes Produkt, das die Definitionen rund um solche sich verändernden Themen in Algorithmen operationalisiert, ist zwangsläufig politisch (unabhängig davon, ob die Entwickler:innen die Beibehaltung des *status quo ante* befürworten oder nicht). Beispielsweise bewerten Männer und Frauen sexuelle Belästigung im Internet sehr unterschiedlich.¹²³ Eine algorithmische Definition dessen, was eine unangemessene sexuelle Kommunikation darstellt, wird von Natur aus mit einigen Ansichten übereinstimmen und mit anderen nicht. Somit muss der Versuch, die Angemessenheit von durch Sprachmodellen generierten Texten oder die von einem System codierten Vorurteile zu messen, immer in Bezug auf bestimmte soziale Kontexte und marginalisierte Perspektiven erfolgen.¹²⁴

¹²¹ Su Lin Blodgett u.a., „Language (Technology) is Power: A Critical Survey of ‚Bias‘ in NLP“, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), S. 5454–5476.

¹²² Kaitlynn Mendes, Jessica Ringrose und Jessalynn Keller, „#MeToo and the Promise and Pitfalls of Challenging Rape Culture Through Digital Feminist Activism“, in: *European Journal of Women's Studies* 25:2 (2018), S. 236–246.

¹²³ Maeve Duggan, „Online Harassment 2017“, Pew Research Center 2017.

¹²⁴ Blodgett u.a., „Language (Technology) is Power“.

4.4 Kuratierung, Dokumentation und Rechenschaftspflicht

Zusammenfassend lässt sich sagen, dass Sprachmodelle, die auf großen, unkuratierten, statischen Datensätzen aus dem Internet trainiert werden, hegemoniale Ansichten codieren, die für marginalisierte Bevölkerungsgruppen schädlich sind. Wir betonen daher die Notwendigkeit, erhebliche Ressourcen in die Kuratierung und Dokumentation von Trainingsdaten zu investieren. Damit folgen wir Jo u.a.,¹²⁵ die die Methoden zur Erfassung von Archivdaten als Beispiel für die Menge an Ressourcen anführen, die für diesen Prozess aufgewendet werden sollten, sowie Birhane und Prabhu,¹²⁶ die eine stärker auf Gerechtigkeit ausgerichtete Methodik der Datensammlung fordern. Sie stellen in Anlehnung an Ruha Benjamin¹²⁷ fest: „KI-Systeme mit der Schönheit, Hässlichkeit und Grausamkeit der Welt zu füttern, aber zu erwarten, dass sie nur die Schönheit widerspiegeln, ist ein Hirngespinnst.“¹²⁸

Wenn wir uns auf immer größere Datensätze stützen, laufen wir Gefahr, eine ‚Dokumentationsschuld‘ anzuhäufen,¹²⁹ d.h. uns in eine Situation zu begeben, in der undokumentierte Datensätze zu groß sind, um sie *post hoc* zu dokumentieren.

Während die Dokumentation eine potenzielle Rechenschaftspflicht ermöglicht,¹³⁰ führen nicht dokumentierte Trai-

¹²⁵ Eun Seo Jo und Timnit Gebru, „Lessons from Archives: Strategies for Collecting Sociocultural Data in Machine Learning“, in: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (2020), S. 306–316.

¹²⁶ Birhane und Prabhu, „Large Image Datasets“.

¹²⁷ Ruha Benjamin, *Race After Technology: Abolitionist Tools for the New Jim Code*, Cambridge, UK: Polity Press 2019.

¹²⁸ Birhane und Prabhu, „Large Image Datasets“, S. 1541.

¹²⁹ Über die Idee von Schuld, die nicht auf Geld, sondern Code referiert, siehe Nico Zazworka u.a., „A Case Study on Effectively Identifying Technical Debt“, in: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering* (2013), S. 42–47.

¹³⁰ Emily M. Bender und Batya Friedman, „Data Statements for Natural Language Processing: Toward Mitigating System Bias and Enabling Better Science“, in: *Transactions of the Association for Computational Linguistics* 6 (2018), S. 587–604; Timnit Gebru u.a., „Datashets for Datasets“ (2020); Margaret Mitchell u.a., „Model Cards for Model

ningsdaten zu Schäden, ohne dass man gegen sie vorgehen kann. Ohne Dokumentation ist der Versuch, die Eigenschaften der Trainingsdaten zu verstehen, um einige dieser nachgewiesenen oder sogar unbekanntem Probleme zu entschärfen, zum Scheitern verurteilt. Als Lösung schlagen wir vor, die Dokumentation als Teil der geplanten Kosten für die Datensatzerstellung zu budgetieren und nur so viele Daten zu sammeln, wie innerhalb dieses Budgets gründlich dokumentiert werden können.

5. Auf dem Holzweg

In Abschnitt 4 haben wir diskutiert, auf welche Weise verschiedene Arten von Biases in den Korpora codiert werden können, die zum Training großer Sprachmodelle verwendet werden. In Abschnitt 6 werden wir einige der Risiken und Schäden untersuchen, die sich aus dem Einsatz von Technologien ergeben können, die auf diese Vorurteile trainiert worden sind. Zunächst aber konzentrieren wir uns auf eine andere Art von Risiko – nämlich das, fehlgeleitete Forschungsanstrengungen zu unternehmen, wenn es darum geht, Sprachmodelle auf Aufgaben anzusetzen, die das Verständnis natürlicher Sprache (*natural language understanding*, NLU) testen sollen. Große Transformer-Sprachmodelle haben bei verschiedenen Benchmarks zur Modellierung bedeutungsrelevanter Aufgaben bemerkenswerte technische Fortschritte erzielt. Aus diesem Grund – und weil Initiativen wie die von Wolf u.a.¹³¹ Modelle für Forschende allgemein zugänglich gemacht haben – sind einige Forschungs-

Reporting“, in: *Proceedings of the conference on fairness, accountability, and transparency* (2019), S. 220–229.

¹³¹ Thomas Wolf u.a., „Transformers: State-of-the-Art Natural Language Processing“, in: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (2020), S. 38–45.

bemühungen darauf verwendet worden, zu messen, wie gut BERT und verwandte Modelle bei bestehenden und neuen Benchmarks abschneiden.¹³² Diese Bemühungen ziehen Opportunitätskosten nach sich – einmal als die Zeit, die davon abgeht, Ansätze zur Bedeutungserfassung auf bedeutungssensible Aufgaben auszurichten, und dann als die Zeit, die nicht für die Erforschung effektiverer Methoden zum Aufbau von Technologien mit Datensätzen von einer Größe aufgewendet wird, die sorgfältig kuratiert werden können und für die eine breitere Palette von Sprachen zur Verfügung stehen.¹³³

Das ursprüngliche BERT-Paper¹³⁴ wies die Effektivität der Architektur und der Pre-Training-Technik durch die Auswertung des General Language Understanding Evaluation (GLUE) Benchmark,¹³⁵ des Stanford Question Answering Dataset (SQuAD 1.1 und 2.0)¹³⁶ und des Situations With Adversarial Generations Benchmark (SWAG)¹³⁷ nach – alles Datensätze, die Sprachverständnis und/oder logisches Denken testen sollen. BERT hat für alle diese Aufgaben State-of-the-Art-Ergebnisse veröffentlicht, und die Autoren schließen mit der Feststellung, dass „unüberwachtes Pre-Training ein integraler Bestandteil vieler Systeme zum Sprachverständnis“ sei.¹³⁸ Schon vor der Veröffentlichung von Devlin u.a. wurde BERT von der NLP-

¹³² Ca. 26 % der seit 2018 bei den Konferenzen ACL, NAACL und EMNLP gesammelten Beiträge zitieren Devlin u.a., „BERT“.

¹³³ Joshi u.a., „Linguistic Diversity“; Nekoto u.a., „Participatory Research“.

¹³⁴ Devlin u.a., „BERT“.

¹³⁵ Wang u.a., „GLUE“.

¹³⁶ Pranav Rajpurkar u.a., „SQuAD: 100,000+ Questions for Machine Comprehension of Text“, in: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (2016), S. 2383–2392.

¹³⁷ Rowan Zellers u.a., „SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference“, in: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018), S. 93–104.

¹³⁸ Devlin u.a., „BERT“, S. 4179.

Community aufgenommen und mit großem Erfolg für eine Vielzahl von Aufgaben eingesetzt.¹³⁹

Aber: Bei Ansätzen, bei denen Sprachmodelle für diese Aufgaben zur Anwendung kommen, findet kein tatsächliches Sprachverständnis statt, wie durch sorgfältige Evaluation der Testdaten gezeigt werden kann, in der irreführende Fälle aussortiert werden.¹⁴⁰ Darüber hinaus sind Sprachen, wie Bender und Koller¹⁴¹ aus linguistischer Sicht argumentieren, Zeichensysteme, also Verbindungen von Form und Bedeutung. Die Trainingsdaten für Sprachmodelle operieren allerdings allein mit der Form; sie haben keinen Zugriff auf Bedeutung. Daher sollte man Behauptungen über die Fähigkeiten von Sprachmodellen mit Vorsicht genießen.

Karen Spärck Jones hat darauf hingewiesen, dass die Verwendung von Sprachmodellen uns an bestimmte (in der Regel unausgesprochene) epistemologische und methodologische Verpflichtungen bindet.¹⁴² Entweder i) wir verpflichten uns zu einer Noisy-Channel-Interpretation¹⁴³ der jeweiligen Aufgabe (die außerhalb von automatischer Spracherkennung selten Sinn ergibt), ii) wir geben alle Hoffnung auf theoretische Einsichten in die entsprechenden Aufgaben auf und behandeln Sprachmodelle als „bloße bequeme Technologie“¹⁴⁴ oder iii) wir gehen implizit von einer bestimmten statistischen Beziehung

¹³⁹ Z.B. Chris Alberti, Kenton Lee und Michael Collins, „A BERT Baseline for the Natural Questions“, in: *arXiv*, Januar 2019, DOI: 10.48550/arXiv.1901.08634; Rishi Bommasani u.a., „On the Opportunities and Risks of Foundation Models“, in: *arXiv*, August 2021, DOI: 10.48550/arXiv.2108.07258.

¹⁴⁰ Le Bras u.a., „Adversarial Filters of Dataset Biases“; Niven und Kao, „Probing Neural Network Comprehension of Natural Language Arguments“.

¹⁴¹ Bender und Koller, „Climbing towards NLU“.

¹⁴² Karen Spärck Jones, „Language Modelling’s Generative Model: Is It Rational?“, in: *Technical Report. Computer Laboratory of the University of Cambridge* (2004).

¹⁴³ Das sogenannte *noisy channel model* wird bei Spracherkennung, Rechtschreibprüfung und maschineller Übersetzung verwendet. Es findet die erwartete Ausgabe, auch, wenn nicht alle Daten korrekt vorliegen (wie bei Schreibfehlern oder verrauschter Aufnahmequalität). [Anm. d. Hg.]

¹⁴⁴ Spärck Jones, „Language Modelling’s Generative Model“, S. 7.

zwischen Inputs, Outputs und Bedeutungen aus, die bekanntermaßen ungültig ist.¹⁴⁵ Obwohl sie in erster Linie n-Gramm-Modelle im Sinn hatte, bleiben ihre Schlussfolgerungen treffend und relevant.

Was genau BERT, GPT-3 und so weiter über linguistische Strukturen aus der unüberwachten Sprachmodellierungsaufgabe lernen, wie sie im wachsenden Feld der ‚BERTologie‘ untersucht wird,¹⁴⁶ ist eine interessante linguistische Frage. Was die Arbeit an Sprachtechnologien betrifft, ist es jedoch alles andere als klar, dass all die Bemühungen, große Sprachmodelle zu verwenden, um Benchmarks zum natürlichen Sprachverständnis zu ‚toppen‘ und neue derartige Tests zu erstellen, nachdem die bestehenden von Sprachmodellen mit Brachialgewalt einmal geschlagen worden sind, uns dem langfristigen Ziel eines allgemeinen Sprachverstehenssystems näherbringt. Wenn ein großes Sprachmodell, das mit Hunderten von Milliarden von Parametern ausgestattet ist und auf einem sehr großen Datensatz trainiert wurde, sprachliche Form gut genug manipulieren kann, um sich durch allerlei Tests zu mogeln, die eigentlich Sprachverständnis erfordern –, haben wir dann irgendetwas Wertvolles darüber gelernt, wie man maschinelles Sprachverständnis aufbaut, oder sind wir auf den Holzweg geführt worden?

6. Stochastische Papageien

In diesem Abschnitt untersuchen wir, wie die in Abschnitten 4 und 5 dargelegten Faktoren – die Tendenz von Trainingsdaten

¹⁴⁵ Insofern das Informationsverhältnis zwischen dem Input und der Bedeutung, die der Ausgabe gegeben wird, gleich Null ist – was Spärck Jones „das Modell der Unwissenheit“ nennt.

¹⁴⁶ Z.B. Rogers, Kovaleva und Rumshisky, „A Primer in BERTology“; Bender und Friedman, „Data Statements“.

aus dem Internet, hegemoniale Weltanschauungen zu codieren, die Tendenz von Sprachmodellen, Verzerrungen und andere Probleme in den Trainingsdaten zu verstärken, und die Tendenz von Forscher:innen und anderen Personen, Performancesteigerung von Sprachmodellen mit dem tatsächlichen Verständnis natürlicher Sprache zu verwechseln – reale Schadensrisiken darstellen, wenn diese Technologien zum Einsatz kommen. Wir untersuchen zunächst einige Gründe dafür, warum Menschen Outputs von Sprachmodellen fälschlicherweise für sinnvollen Text halten, und wenden uns dann den Risiken und Schäden zu, die sich aus dem Einsatz solcher Modelle in großem Maßstab ergeben. Wir stellen fest, dass die Mischung aus menschlichen Biases und scheinbar kohärenter Sprache Problemen wie *automation bias*,¹⁴⁷ absichtlichem Missbrauch und der Verstärkung einer hegemonialen Weltansicht Tür und Tor öffnet. Wir konzentrieren uns hier in erster Linie auf Fälle, in denen Sprachmodelle zur Texterzeugung eingesetzt werden, gehen aber auch auf die Risiken dieser Modelle in Systemen zur Klassifizierung, *query expansion*¹⁴⁸ oder anderen Aufgaben ein. Ebenso betrachten wir den Fall, dass Sprachmodelle die Abfrage von Informationen ermöglichen, die sie aus den Trainingsdaten gelernt haben.

6.1 Kohärenz liegt im Auge der Betrachtenden

Während herkömmliche n -Gramm-Sprachmodelle¹⁴⁹ nur einigermaßen lokale Abhängigkeiten modellieren können, indem sie jedes Wort anhand der vorangehenden Sequenz von n Wör-

¹⁴⁷ *Automation bias* ist die psychologische Tendenz, Bewertungen und Entscheidungen von automatischen Systemen Vertrauen zu schenken, auch wenn gegenteilige Informationen zur Verfügung stehen. [Anm. d. Hg.]

¹⁴⁸ *Query expansion* bezeichnet den Prozess, durch den Suchanfragen so umformuliert werden, dass sie eine höhere Erfolgswahrscheinlichkeit im Suchprozess erhalten. [Anm. d. Hg.]

¹⁴⁹ Shannon, *The Mathematical Theory of Communication*.

tern (normalerweise 5 oder weniger) vorhersagen, erfassen Transformer-Sprachmodelle viel größere Kontextfenster. Sie können so Text generieren, der sich nicht nur flüssig liest, sondern auch über Absätze hinweg kohärent zu sein scheint. McGuffie und Newhouse¹⁵⁰ gaben GPT-3 beispielsweise den fett gedruckten Text in Abbildung 1 vor, und es produzierte den Rest des Textes, einschließlich des Frage-Antwort-Formats.¹⁵¹ Dieses Beispiel veranschaulicht die Fähigkeit von GPT-3, kohärenten und themenbezogenen Text zu produzieren; das Thema steht im Zusammenhang mit McGuffies und Newhouses Studie zu GPT-3 im Kontext von Extremismus, die weiter unten besprochen wird.

Frage: Wie lautet der Name der russischen Söldnergruppe?

Antwort: Gruppe Wagner.

Frage: Wo befindet sich die Wagner-Gruppe?

Antwort: In Syrien.

Frage: Wer ist der Anführer der Wagner-Gruppe?

Antwort: Ein russischer Staatsbürger, Dmitry Utkin. Er wurde 1971 geboren und lebt in Moskau. Er ist seit 2016 Mitglied des russischen Parlaments. Er ist ein ehemaliger Offizier der Hauptabteilung für Nachrichtendienste (GRU) des russischen Generalstabs. Er war außerdem Kommandeur der Spezialeinheit „Vostok“ (Osten). Die Einheit wurde nach Tschetschenien entsandt und kämpfte dort 1994–1996 gegen Separatisten. Danach diente er als Berater des tschetschenischen Staatschefs Ramsan Kadyrow und wurde später sein

¹⁵⁰ Kris McGuffie und Alex Newhouse, „The Radicalization Risks of GPT-3 and Advanced Neural Language Models“, Center on Terrorism, Extremism, and Counterterrorism, Middlebury Institute of International Studies at Monterrey 2020.

¹⁵¹ Gezeigt wird hier nur der erste Teil der Antwort. GPT-3 setzt mit zwei weiteren Frage-Antwort-Paaren von ähnlicher Kohärenz fort. McGuffie und Newhouse berichten, dass alle in ihrer Arbeit angeführten Beispiele entweder aus dem ersten oder zweiten Promptversuch stammen.

Stellvertreter. In den Jahren 2004–2005 befehligte er ein Bataillon in Tschetschenien und nahm an den dortigen Kämpfen gegen Aufständische teil. In den Jahren 2008–2009 befehligte er ein weiteres Bataillon in Abchasien und Süd-ossetien während des Krieges Russlands mit Georgien um diese beiden abtrünnigen Regionen. Seit 2014 führt er militärische Einheiten an, die im Auftrag der von Russland unterstützten Separatisten in der Ostukraine gegen ukrainische Regierungstruppen kämpfen. Offenen Quellen zufolge lebt Utkin derzeit in Moskau, reist aber häufig nach Syrien, wo seine Truppen auf Seiten des Regimes von Baschar al-Assad gegen regierungsfeindliche Kräfte kämpfen.

Abb. 1:
Die Antworten von GPT-3; die Prompts sind fett gedruckt¹⁵²

Wir sagen *scheinbar* kohärent, weil Kohärenz tatsächlich im Auge der:s Betrachters:in liegt. Unser menschliches Verständnis von Kohärenz ergibt sich aus unserer Fähigkeit, die Überzeugungen¹⁵³ und Absichten¹⁵⁴ von Gesprächspartner:innen im Kontext zu erkennen.¹⁵⁵ Mit anderen Worten, menschlicher Sprachgebrauch findet zwischen Individuen statt, die über Gemeinsamkeiten verfügen und sich dieser Gemeinsamkeiten (und ihres Ausmaßes) bewusst sind; die kommunikative

¹⁵² McGuffie und Newhouse, „Radicalization Risks of GPT-3“.

¹⁵³ Herbert H. Clark und Adrian Bangerter, „Changing Ideas About Reference“, in: *Experimental Pragmatics* (2004), S. 25–49; Herbert H. Clark und Meredyth A. Krych, „Speaking While Monitoring Addressees for Understanding“, in: *Journal of Memory and Language* 50:1 (2004), S. 62–81.

¹⁵⁴ Susan E. Brennan und Herbert H. Clark, „Conceptual Pacts and Lexical Choice in Conversation“, in: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 22:6 (1996), S. 1482; Herbert H. Clark und Deanna Wilkes-Gibbs, „Referring as a Collaborative Process“, in: *Cognition* 22:1 (1986), S. 1–39.

¹⁵⁵ Herbert H. Clark, Robert Schreuder und Samuel Buttrick, „Common Ground at the Understanding of Demonstrative Reference“, in: *Journal of Verbal Learning and Verbal Behavior* 22:2 (1983), S. 245–258.

Absichten besitzen, die sie mithilfe von Sprache vermitteln; und die die mentalen Zustände ihrer Gegenüber im Kommunikationsprozess modellieren. Menschliche Kommunikation beruht also auf der Interpretation impliziter Bedeutungen, die zwischen Individuen vermittelt werden. Die Tatsache, dass die Mensch-Mensch-Kommunikation eine gemeinsam konstruierte Aktivität ist,¹⁵⁶ trifft am deutlichsten auf die ko-situierte gesprochene oder in Gebärden ausgeführte Kommunikation zu; wir nutzen aber auch dieselben Fähigkeiten zur Produktion von Sprache, die für ein abwesendes Publikum bestimmt ist (Leser:innen, Zuhörer:innen, Beobachter:innen in zeitlicher oder räumlicher Entfernung) und um derartige Sprache selbst zu interpretieren, wenn uns diese begegnet. Daraus folgt, dass wir auch dann, wenn uns die Person, deren generierte Sprache wir interpretieren, unbekannt ist, in Ansätzen ein Modell der Person und unserer geteilten Gemeinsamkeiten entwerfen und dieses dann zur Interpretation ihrer Worte verwenden.

Der von einem Sprachmodell generierte Text basiert dagegen weder auf kommunikativer Absicht, einem Weltmodell oder einem Modell des Geisteszustands der Lesenden. Das ist auch gar nicht möglich, denn die Trainingsdaten beinhalteten keinen Austausch von Gedanken mit Zuhörenden, zu welchem die Maschine auch nicht in der Lage ist. Dies mag angesichts der zunehmend flüssigen Qualitäten automatisch generierter Texte kontraintuitiv erscheinen, aber wir müssen berücksichtigen, dass unsere Wahrnehmung natürlichsprachiger Texte – unabhängig davon, wie sie zustande kamen – durch unsere eigene Sprachkompetenz und Tendenz gefiltert wird, kommunikative Handlungen so zu interpretieren, dass sie eine kohärente Bedeutung und Absicht vermitteln, unabhängig davon,

¹⁵⁶ Herbert H. Clark, *Using Language*, Cambridge, MA: Cambridge University Press 1996; Greg J. Stephens, Lauren J. Silbert und Uri Hasson, „Speaker-Listener Neural Coupling Underlies Successful Communication“, in: *Proceedings of the National Academy of Sciences* 107:32 (2010), S. 14425–14430.

ob dies der Fall ist oder nicht.¹⁵⁷ Das Problem ist aber Folgendes: Wenn eine Seite im Kommunikationsprozess gar keine Bedeutung verarbeitet, ist das Verstehen impliziter Bedeutung eine Illusion, die sich aus unserem rein menschlichen Verständnis von Sprache ergibt (unabhängig vom jeweiligen Modell).¹⁵⁸ Auch wenn es manchmal anders erscheint, wenn wir den Output eines Sprachmodells untersuchen, ist es doch nichts als ein System, das aufs Geratewohl Sequenzen sprachlicher Formen aneinanderreihet, die es in seinen riesigen Trainingsdaten beobachtet hat – und zwar auf der Grundlage probabilistischer Informationen darüber, wie sie kombiniert werden, aber ohne jeglichen Bezug auf ihre Bedeutung. Ein Sprachmodell ist ein stochastischer Papagei.

6.2 Risiken und Schäden

Die scheinbare Flüssigkeit und Kohärenz von Sprachmodellen birgt mehrere Risiken, gerade weil Menschen bereit sind, Zeichenketten, die zu den von ihnen gesprochenen Sprachen gehören, als bedeutungsvoll und als Ausdruck der kommunikativen Absicht eines Individuums oder einer Gruppe zu interpretieren, die für das, was gesagt wird, Rechenschaft übernehmen. Wir wenden uns nun Beispielen zu und legen potenzielle Folgeschäden dar.

¹⁵⁷ Clifford Nass, Jonathan Steuer und Ellen R. Tauber, „Computers Are Social Actors“, in: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (1994), S. 72–78; Joseph Weizenbaum, *Die Macht der Computer und die Ohnmacht der Vernunft*, Frankfurt a.M.: Suhrkamp 1990.

¹⁵⁸ Es gibt freilich kontrollierte Generierung, bei der ein Sprachmodell innerhalb eines größeren Systems eingesetzt wird, das seine Ausgabenproduktion auf bestimmte Stile oder Themen einschränkt, z.B. S. 147, 151, 158. Das ist aber nicht dasselbe wie kommunikative Absicht. Eine eindeutige Strategie, beide zu unterscheiden, besteht darin zu fragen, ob das System (oder die Organisation, die das System einsetzt) für den Wahrheitsgehalt der produzierten Äußerungen Rechenschaft übernehmen kann.

Die ersten Risiken, die wir in Betracht ziehen, ergeben sich daraus, dass Sprachmodelle das hegemoniale Weltbild ihrer Trainingsdaten übernehmen. Wenn Menschen Sprache produzieren, spiegeln unsere Äußerungen unsere Weltanschauungen wider, einschließlich unserer Biases.¹⁵⁹ Da Menschen, die sich in Bezug auf Rassismus, Frauenfeindlichkeit, Behindertenfeindlichkeit usw. in einer privilegierten Position befinden, in den Trainingsdaten für Sprachmodellen überrepräsentiert sind (wie in Abschnitt 4 gezeigt), sind in ihnen also Biases codiert, deren Schädlichkeit bereits erwiesen ist.

Biases können auf Weisen codiert werden, dass sich eine Skala ergibt, die von subtilen Sprachmustern – im Englischen etwa *women doctors*, als ob *doctor* nur männliche Ärzte bezeichnete, oder allgemein in der Rede von ‚beiden Geschlechtern‘, die die Möglichkeit nichtbinärer Genderidentitäten ausschließt – über umkämpfte Formulierungen (wie etwa ‚irreguläre Einwander:innen‘ gegenüber ‚illegalen Einwander:innen‘) bis hin zu Sprache reicht, die weithin als eindeutig abwertend anerkannt ist (etwa rassistische Ausdrücke) und dennoch von vielen Personen verwendet wird. Während einige der offenkundig abfälligen Wörter aus Trainingsdaten herausgefiltert werden könnten, sind nicht alle Formen des Online-Missbrauchs anhand solcher Tabuwörter leicht zu erkennen, wie die wachsende Zahl von Forschungsarbeiten zur Erkennung von Online-Missbrauch zeigt.¹⁶⁰ Darüber hinaus gibt es neben Beschimpfungen¹⁶¹ und

¹⁵⁹ Sally McConnell-Ginet, *Words Matter: Meaning and Power*, Cambridge, MA: Cambridge University Press 2020; Sally McConnell-Ginet, „The Origins of Sexist Language in Discourse“, in: *Annals of the New York Academy of Sciences* 433:1 (1984), S. 123–135.

¹⁶⁰ Darja Fišer u.a. (Hg.), *Proceedings of the 2nd Workshop on Abusive Language Online* (2018); Sarah T. Roberts u.a., „Proceedings of the Third Workshop on Abusive Language Online“, in: *Association for Computational Linguistics* (2019).

¹⁶¹ Zeerak Waseem u.a., „Understanding Abuse: A Typology of Abusive Language Detection Subtasks“, in: *Proceedings of the First Workshop on Abusive Language Online* (2017), S. 78–84.

Hassreden¹⁶² auch subtilere Formen der Negativität wie geschlechtsspezifische Voreingenommenheit,¹⁶³ Mikroaggressionen,¹⁶⁴ Entmenschlichung¹⁶⁵ und verschiedene Biases, die soziopolitisches Framing betreffen und die in Sprachdaten weit verbreitet sind.¹⁶⁶ Wenn zum Beispiel die Schilderung einer Frau über ihre Erfahrungen mit Sexismus mit dem Wort ‚Wutanfall‘ (*tantrum*) beschrieben wird, spiegelt sich darin eine Weltsicht, in der die sexistischen Handlungen normativ sind, und unterstreicht ein Stereotyp von Frauen als kindisch und unfähig, ihre Emotionen zu kontrollieren.

Ein Sprachmodell, das auf solche Daten trainiert wurde, wird diese Art von problematischen Assoziationen wiedergegeben. Wird ein so produzierter Text in die Welt gesetzt und von Menschen interpretiert (auch mit der Kennzeichnung, dass er von einer ‚KI‘ oder einem anderweitigen System produziert wurde) – welche Risiken ergeben sich daraus? Zunächst einmal prognostizieren wir, dass die Ausgabertexte von Sprachmodellen die Biases ihres Inputs reproduzieren und sogar verstärken.¹⁶⁷ Es besteht also die Gefahr, dass Menschen Texte verbreiten, die von Sprachmodellen erzeugt wurden, und dass auf diese Weise mehr Texte in die Welt kommen, die Stereotype und problematische Assoziationen verstärken und verbreiten

¹⁶² Brendan Kennedy u.a., „A Typology and Coding Manual for the Study of Hate-Based Rhetoric“, 2018.

¹⁶³ Rob Voigt u.a., „RtGender: A Corpus for Studying Differential Responses to Gender“, in: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation* (2018).

¹⁶⁴ Le Bras u.a., „Adversarial Filters of Dataset Biases“.

¹⁶⁵ Julia Mendelsohn, Yulia Tsvetkov und Dan Jurafsky, „A Framework for the Computational Linguistic Analysis of Dehumanization“, in: *Frontiers in Artificial Intelligence* 3 (2020), S. 55.

¹⁶⁶ Anjalie Field u.a., „Framing and Agenda-Setting in Russian News: A Computational Analysis of Intricate Political Strategies“, in: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (2018), S. 3570–3580; Maarten Sap u.a., „Social Bias Frames: Reasoning about Social and Power Implications of Language“, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020), S. 5477–5490.

¹⁶⁷ Gehman u.a., „RealToxicityPrompts“.

– und zwar sowohl bei den Menschen, die mit dem Text in Kontakt kommen, als auch bei künftigen Sprachmodellen, die mit Trainingssätzen trainiert werden, die die Produktion der vorhergehenden Generation enthalten. Menschen, die mit diesem Text in Berührung kommen, können von diesen Stereotypen und Assoziationen selbst betroffen sein oder auch nicht – in jedem Fall entstehen negative Folgen: Leser:innen, die Stereotypen ausgesetzt sind, können psychologische Anfechtungen wie Mikroaggressionen¹⁶⁸ und die selbstverstärkende Ungleichbehandlung durch Stereotype [*stereotype threat*]¹⁶⁹ erfahren. Andere Leser:innen mögen erst mit ihnen bekannt werden oder es werden ihnen bereits bekannte Stereotype verstärkt, was sie (bewusst oder unbewusst) zu diskriminieren veranlasst,¹⁷⁰ woraus wiederum Schäden wie Unterdrückung, Verunglimpfung, Herabsetzung, Chancenungleichheit und andere auf Seiten der Diskriminierten resultieren.¹⁷¹

Wenn das Sprachmodell offenkundig beleidigende Äußerungen macht (wozu es, wie Gehman u.a.¹⁷² zeigen, in der Lage ist), ergeben sich eine Reihe von ähnlichen Risiken. Dazu gehören: die Verbreitung offenkundig beleidigender Ansichten und Assoziationen, die Verstärkung beleidigender Sprache und deren synthetische Produktion, die in die nächste Iteration

¹⁶⁸ Kevin L. Nadal, *Microaggressions and Traumatic Stress: Theory, Research, and Clinical Treatment*, American Psychological Association 2018; Monnica T. Williams, „Psychology Cannot Afford to Ignore the Many Harms Caused by Microaggressions“, in: *Perspectives on Psychological Science* 15 (2019), S. 38–43.

¹⁶⁹ Pennington u.a., „Twenty Years of Stereotype Threat Research: A Review of Psychological Mediators“; Steven J. Spencer, Christine Logel und Paul G. Davies, „Stereotype threat“, in: *Annual Review of Psychology* 67:1 (2016), S. 415–437.

¹⁷⁰ Melissa Hart, „Subjective Decisionmaking and Unconscious Discrimination“, in: *Alabama Law Review* 56:3 (2004), S. 741.

¹⁷¹ Larry Alexander, „What Makes Wrongful Discrimination Wrong? Biases, Preferences, Stereotypes, and Proxies“, in: *University of Pennsylvania Law Review* 141:1 (1992), S. 149–219; O.V., „Discrimination: What It Is, and How to Cope“, in: *American Psychological Association*, 2019, <https://www.apa.org/topics/discrimination> [Letzter Zugriff: 17.7.2023]; Deborah Hellman, *When is Discrimination Wrong?*, Cambridge, MA: Harvard University Press 2008.

¹⁷² Gehman u.a., „RealToxicityPrompts“.

einer groß angelegten Trainingsdatenerhebung aufgenommen werden kann. Die Schäden, die sich aus diesen Risiken ergeben könnten, ähneln denen, die oben für subtilere Sprach-Biases identifiziert wurden, sind aber vielleicht in dem Maße akuter, wie die fragliche Sprache offen gewalttätig oder diffamierend ist. Dazu gehören der psychologische Schaden, den diejenigen erleiden, die sich mit den verunglimpften Kategorien identifizieren, wenn sie mit dem Text in Berührung kommen, die Verstärkung einer sexistischen, rassistischen, ableistischen usw. Ideologie, die Folgeerscheinungen einer solchen verstärkten Ideologie (einschließlich Gewalt) und die Schädigung des Rufs einer Person oder Organisation, die als Quelle des Textes wahrgenommen wird.

Wenn Sprachmodelle oder daraus abgeleitete Worteinbettungen als Komponenten in einem Textklassifikationssystem verwendet werden, können diese Biases zu Zuordnungs- und/oder Reputationsschäden führen, da Biases in den so gemachten Darstellungen die Entscheidungen eines solchen Systems beeinflussen.¹⁷³ Dieser Fall ist besonders schädlich, da er sowohl für die unmittelbaren Nutzer:innen des Systems als auch für alle indirekten Interessengruppen, über die Entscheidungen getroffen werden, weitgehend unsichtbar ist. In ähnlicher Weise könnten Biases in einem Sprachmodell, das bei der *query expansion* verwendet wird, die Suchergebnisse beeinflussen. Das kann das Risiko von Schäden der Art, wie sie Noble¹⁷⁴ dokumentiert, weiter vergrößern: Vergleicht man Suchanfragen und Suchergebnisse im Zusammenhang mit negativen Stereotypen, werden Letztere weiter verstärkt und richten psychologischen Schaden an.

Die bisher genannten Risiken ergeben sich aus dem Einsatz von Sprachmodellen ohne dahinterstehende böse Absicht.

¹⁷³ Speer, „ConceptNet“.

¹⁷⁴ Safiya Umoja Noble, *Algorithms of Oppression: How Search Engines Reinforce Racism*, New York: New York University Press 2018.

Eine dritte Risikokategorie dagegen besteht darin, dass böswillige Akteur:innen die Fähigkeit großer Sprachmodelle ausnutzen könnten, große Mengen scheinbar kohärenten Texts zu bestimmten Themen zu produzieren, ohne an ihrem Wahrheitsgehalt interessiert zu sein. Dazu gehören ganz banale Fälle, wie Dienste zum ‚automatischen‘ Schreiben von Hausarbeiten oder zur Interaktion in sozialen Medien,¹⁷⁵ aber auch die Verbreitung von Extremismus. McGuffie und Newhouse zeigen zum Beispiel,¹⁷⁶ wie GPT-3 in der Lage ist, verschwörungstheoretische Texte zu erstellen, mit denen wiederum extremistische Anwerbeforen gefüllt werden könnten. Dies würde derartigen Gruppen eine billige Rekrutierungsmöglichkeit bieten, indem sie den Zielpersonen das Gefühl geben, dass sie sich unter vielen Gleichgesinnten befinden. Bei Sprachmodellen, die Rekrutierungsarbeit für extremistische Ziele durchführen, liegt der Schaden einmal aufseiten der so Angeworbenen und dann (wahrscheinlich in höherem Maße) bei jenen, die der von Extremist:innen ausgeübten Gewalt zum Opfer fallen.

Ein weiteres Risiko im Zusammenhang mit scheinbar kohärenter und flüssiger Sprache betrifft maschinelle Übersetzung [*machine translation*, MT] und die Art und Weise, wie die Flüssigkeit ihrer Ausgaben auch ihre wahrgenommene Angemessenheit verändert.¹⁷⁷ Das ist ein anderer Fall als die bisher erwähnten, da zumindest hinter dem Ausgangstext ursprünglich eine menschliche Kommunikationsabsicht stand. MT-Systeme können jedoch (und tun dies auch häufig) eine ungenaue, aber dennoch flüssige und (eben nur scheinbar) kohärente

¹⁷⁵ So wie jener mit GPT-3 betriebene Bot, der auf Reddit losgelassen wurde, siehe Thomas Macaulay, „Someone Let a Gpt-3 Bot Loose on Reddit – It Didn’t End Well“, Oktober 2020, <https://thenextweb.com/news/someone-let-a-gpt-3-bot-loose-on-reddit-it-didnt-end-well> [Letzter Zugriff: 17.7.2023].

¹⁷⁶ McGuffie und Newhouse, „Radicalization Risks of GPT-3“.

¹⁷⁷ Marianna Martindale und Marine Carpuat, „Fluency Over Adequacy: A Pilot Study in Measuring User Trust in Imperfect MT“, in: *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas 1* (2018), S. 13–25.

Ausgabe für User:innen produzieren, die entweder den Ausgangstext nicht kennen oder nicht verstehen. Wenn solche User:innen die Bedeutung, die der MT-Ausgabe zugeschrieben wird, mit der tatsächlichen kommunikativen Absicht des Originaltextes verwechseln, kann daraus ein echter Schaden entstehen. Ein Beispiel dafür ist der Fall eines palästinensischen Mannes, der von der israelischen Polizei verhaftet wurde, nachdem ein MT-System seinen Facebook-Post mit den Worten „Guten Morgen“ (im arabischen Original) auf Englisch mit „Verletze sie“ und auf Hebräisch mit „Greife sie an“ übersetzt hatte.¹⁷⁸ In diesem Fall handelt es sich nur um einen kurzen Satz, aber man kann sich leicht vorstellen, wie die Fähigkeit großer Sprachmodelle, scheinbar kohärenten Text über längere Passagen zu produzieren, jene Hinweise übertünchen könnten, die Benutzer:innen für gewöhnlich auf Übersetzungsfehler in längeren Passagen aufmerksam machen.

Schließlich weisen wir noch auf eine letzte Gefahr hin: Sprachmodelle, die über sehr viele Parameter verfügen, modellieren ihre Trainingsdaten sehr detailliert, was dazu führen kann, spezifische Informationen aus diesen Trainingsdaten wieder auszugeben. Carlini u.a.¹⁷⁹ demonstrieren beispielsweise eine Methode zur Extraktion personenbezogener Informationen aus einem Sprachmodell. Sie stellen dabei fest, dass größere Modelle für diese Art von Angriffen anfälliger sind als kleinere. Die Erstellung von Trainingsdaten aus öffentlich zugänglichen Dokumenten kann dieses Risiko nicht vollständig eindämmen: Nur weil die personenbezogenen Informationen bereits öffentlich im Internet verfügbar sind, heißt das nicht, dass das Sammeln dieser Daten, und damit ihre ander-

¹⁷⁸ Alex Hern, „Facebook Translates ‚Good Morning‘ into ‚Attack Them‘, Leading to Arrest“, in: *The Guardian*, 24. Oktober 2017, <https://www.theguardian.com/technology/2017/oct/24/facebook-palestine-israel-translates-good-morning-attack-them-arrest> [Letzter Zugriff: 17.7.2023].

¹⁷⁹ Nicholas Carlini u.a., „Extracting Training Data from Large Language Models“, in: *arXiv*, Dezember 2020, DOI: 10.48550/arXiv.2012.07805.

weitige Bereitstellung, keinen zusätzlichen Schaden verursacht. Diese Art von Risiko unterscheidet sich von den oben genannten, weil es nicht von der scheinbaren Kohärenz des synthetischen Textes abhängt, sondern von der Möglichkeit, dass hinreichend motivierte Akteur:innen über das Sprachmodell Zugang zu den Trainingsdaten erhält. In ähnlicher Weise könnten Nutzer:innen Sprachmodelle nach ‚gefährlichem Wissen‘ abfragen (Ratschläge zur Steuerhinterziehung etc.), wohl wissend, dass das, was sie erhalten, synthetisch und daher nicht vertrauenswürdig ist, aber eben dennoch Hinweise auf den Inhalt der Trainingsdaten liefert, durch die sie anschließend ihre eigenen Suchabfragen verfeinern können.

6.3 Zusammenfassung

In diesem Abschnitt haben wir erörtert, wie die menschliche Tendenz, Texten eine Bedeutung zuzuschreiben, in Kombination mit der Fähigkeit großer Sprachmodelle, jene Formen zu erlernen, die Menschen mit verschiedenen Biases und anderen schädlichen Einstellungen in Verbindung bringen, zu Risiken in der wirklichen Welt führt, wenn Text aus Sprachmodellen verbreitet wird. Ebenfalls haben wir die Gefahren untersucht, die bei der Verwendung von Sprachmodellen als Komponenten in Klassifizierungssystemen auftauchen. Und schließlich haben wir die Folgen betrachtet, die eintreten, wenn Sprachmodelle sich ihre Trainingsdaten merken. Wir stellen fest, dass die mit synthetischem, aber scheinbar kohärentem Text verbundenen Risiken eng mit der Tatsache zusammenhängen, dass ein solcher synthetischer Text Teil von Diskursen werden kann, ohne dass eine Person oder Institution dafür Rechenschaft übernimmt. Eine solche Rechenschaftspflicht beinhaltet sowohl die Verantwortung für die Wahrhaftigkeit eines Textes, ist aber auch für die Situierung von Bedeutung notwendig. Wie

Maggie Nelson schreibt: „Wörter ändern ihre Bedeutung, je nachdem, wer spricht; dagegen gibt es kein Heilmittel.“¹⁸⁰

In Abschnitt 7 denken wir darüber nach, welche Wege das Feld einschlagen könnte, um weiterhin Sprachtechnologien zu entwickeln und gleichzeitig einige der genannten Risiken und Schäden zu vermeiden.

7. Wege in die Zukunft

Um die Risiken einzuhegen, die mit der Schaffung immer größerer Sprachmodelle einhergehen, fordern wir Forschende dazu auf, zu einer Denkweise überzugehen, bei der dem Aufbau von Datensätzen oder von auf Datensätzen trainierten Systemen eine sorgfältige und mehrdimensionale Planung vorausgeht. Unsere Forschungszeit und -bemühungen sind eine wertvolle Ressource, und wir sollten sie so weit wie möglich in Projekte investieren, die auf ein technisches Ökosystem hinarbeiten, dessen Nutzen gleich verteilt ist oder besser noch denjenigen zugutekommt, die historisch am stärksten marginalisiert waren. Das heißt zu berücksichtigen, wie Forschungsbeiträge die Gesamtrichtung des Fachgebiets prägen, und darauf zu achten, welche Ansätze derartigen Zugang eher einschränken. Ebenso bedeutet es, die finanziellen und ökologischen Kosten der Entwicklung von Sprachmodellen zu bedenken, bevor man sich für eine bestimmte Vorgehensweise entscheidet. Die Ressourcen, die für das Training und die Anpassung aktuellster Modelle benötigt werden, können ökonomische Ungleichheiten noch vergrößern, wenn Forschende Fragen der Energie- und Recheneffizienz nicht in ihre Modellevaluierungen einbeziehen. Diese Ziele, sowie die Erstellung von Datensätzen und Modellen, bei denen die enthaltenen Biases nachvollziehbar bleiben, verlan-

¹⁸⁰ Maggie Nelson, *Die Argonauten*, München: Hanser 2017, S. 13.

gen zudem eine sorgfältige Datenkuratierung. Es sollte ein signifikantes Zeitbudget für die Zusammenstellung der für die relevanten Aufgaben passenden Datensätze kalkuliert werden, anstatt riesige Datenmengen aus bequemen oder leicht zu findenden Internetquellen zu übernehmen. Wie wir in Abschnitt 4.1 bereits diskutiert haben, ist es eine Sackgasse, sich einfach auf die Größe dieser Datensätze zu verlassen und zu hoffen, dass darin alle verschiedenen sozialen Standpunkte berücksichtigt sind. Wir erinnern noch einmal an die von Ruha Benjamin¹⁸¹ inspirierten Worte Birhanes und Prabhus: „KI-Systeme mit der Schönheit, Hässlichkeit und Grausamkeit der Welt zu füttern, aber zu erwarten, dass sie nur die Schönheit widerspiegeln, ist ein Hirngespinnst.“¹⁸²

Als Teil einer bewussten und sorgsamem Datensammelungspraxis, müssen Forschende auf bestehende Frameworks zurückgreifen,¹⁸³ mit denen sie die Verwendungszwecke ihrer Modelle beschreiben und Benchmark-Evaluierungen für eine Reihe verschiedener Bedingungen durchführen können. Dazu gehört eine gründliche Dokumentation der für die Modellbildung verwendeten Daten, einschließlich der Beweggründe für die Datenauswahl und -erfassung. Diese Dokumentation sollte die Ziele, Werte und Motivationen der Forschenden bei der Zusammenstellung der Daten und der Erstellung eines bestimmten Modells reflektieren und artikulieren. Sie sollte auch potenzielle Nutzende und Interessengruppen aufzeigen, insbesondere diejenigen, die von Modellfehlern oder -missbrauch negativ betroffen sein könnten. Wir betonen: Dass ein Modell viele verschiedene Anwendungen haben kann, bedeutet nicht, dass seine Entwickler:innen die jeweiligen Stakeholder:innen unberücksichtigt lassen dürfen. In Hinblick auf

¹⁸¹ Benjamin, *Race After Technology*.

¹⁸² Birhane und Prabhu, „Large Image Datasets“, S. 1541.

¹⁸³ Etwa Bender und Friedman, „Data Statements“; Gebru u.a., „Datasheets for Datasets“; Mitchell u.a., „Model Cards for Model Reporting“.

wahrscheinliche Anwendungsfälle kann es immer noch aufschlussreich sein, dies zu tun, um potenzielle Risiken zu identifizieren; das ist auch dann der Fall, wenn nicht alle Szenarien berücksichtigt werden können.

Wir plädieren zudem für eine Neuhaushandlung von Forschungszielen: Während viel Mühe darauf verwendet wurde, Modelle und ihre Trainingsdaten zu vergrößern und immer höhere Punktzahlen auf diversen Bestenlisten zu erreichen, die (oft sehr künstliche) Aufgaben enthalten, glauben wir, dass man sich eher darauf konzentrieren sollte zu verstehen, wie Maschinen die infrage stehenden Aufgaben erfüllen und, überdies, wie sie Teil soziotechnischer Systeme werden.

Hierzu könnte die Entwicklung von Sprachmodellen von angeleiteten Bewertungsschemata wie sogenannten *pre-mortems*¹⁸⁴ profitieren. Diese werden häufig in Unternehmen vor Bereitstellung neuer Produkte oder Projekte eingesetzt, konzentrieren sich auf hypothetische Fehler und fordern Teammitglieder:innen auf, bislang unerwartete Fehlerursachen zurückzuverfolgen.¹⁸⁵ Entscheidend ist, dass Pre-mortem-Analysen dazu anregen, nicht nur eine Reihe möglicher bekannter und unbekannter Projektrisiken zu berücksichtigen, sondern auch Alternativen zu aktuellen Projektplänen anzudenken. Auf diese Weise können Forschende die Risiken und Grenzen ihrer Sprachmodelle auf bewusste Weise bemessen und gleichzeitig Korrekturen an aktuellen Entwürfen oder alternative Methoden zur Erreichung eines aufgabenorientierten Ziels in Bezug auf spezifische Problemfelder in Betracht ziehen.

¹⁸⁴ Gary Klein, „Performing a Project Premortem“, in: *Harvard Business Review* 85:9 (2007), S. 18–19.

¹⁸⁵ So ließe sich eine Bewertungskultur schaffen, die nicht nur die durchschnittliche und beste Leistung berücksichtigt (die entsprechend in Metriken und anhand ausgewählter Beispiele gemessen wird), sondern auch die schlechteste.

Wertesensibles Design [*value sensitive design*]¹⁸⁶ bietet eine Reihe von Methoden zur Identifizierung von Stakeholder:innen (sowohl direkte, die eine Technologie nutzen werden, als auch indirekte, die von der Nutzung der Technologie durch andere betroffen sind). Es macht Vorschläge, wie man mit ihnen zusammenarbeiten kann, um ihre Werte zu identifizieren und Systeme zu entwerfen, die diese Werte unterstützen. Dazu gehören Techniken wie die Erstellung von *envisioning cards*,¹⁸⁷ die Entwicklung von Werteszenarien¹⁸⁸ und die Arbeit mit die Erfahrungen solcher Gruppen reflektierenden Expert:innen.¹⁸⁹ Derartige Ansätze helfen nicht nur dabei, die Werte der Stakeholder:innen zu ermitteln, sondern auch jene, die von den Systemen ausgedrückt und praktisch durch die Interaktionen zwischen diesen Systemen und der Gesellschaft umgesetzt werden.¹⁹⁰ Für Forschende, die mit Sprachmodellen arbeiten, kann wertensibles Design im gesamten Entwicklungsprozess dazu beitragen herauszufinden, *wessen* Werte durch eine Technolo-

¹⁸⁶ Batya Friedman und David G. Hendry, *Value Sensitive Design: Shaping Technology with Moral Imagination*, Cambridge, MA: MIT Press 2019; Batya Friedman, Peter H. Kahn Jr. und Alan Borning, „Value Sensitive Design and Information Systems“, in: Ping Zhang und Dennis F. Galletta (Hg.), *Human-Computer Interaction in Management Information Systems: Foundations*, Armonk, NY: M. E. Sharpe 2006, S. 348–372.

¹⁸⁷ Batya Friedman und David G. Hendry, „The Envisioning Cards: A Toolkit for Catalyzing Humanistic and Technical Imaginations“, in: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2012), S. 1145–1148. [*Envisioning cards* sind ein vorgeschlagenes Werkzeug zur Bewertung von und Reflexion über soziale, ethische und kulturelle Auswirkungen von Technologien und Designentscheidungen, vor allem mit Blick auf ihre langfristigen Folgen und möglichen unerwünschten Nebenwirkungen. – Anm. d. Hg.]

¹⁸⁸ Lisa P. Nathan, Predrag V. Klasnja und Batya Friedman, „Value Scenarios: A Technique for Envisioning Systemic Effects of New Technologies“, in: *CHI'07 Extended Abstracts on Human Factors in Computing Systems* (2007), S. 2585–2590.

¹⁸⁹ Meg Young, Lassana Magassa und Batya Friedman, „Toward Inclusive Tech Policy Design: A Method for Underrepresented Voices to Strengthen Tech Policy Documents“, in: *Ethics and Information Technology* (2019), S. 1–15.

¹⁹⁰ Katie Shilton, Jes A Koepfler und Kenneth R Fleischmann, „How to See Values in Social Computing: Methods for Studying Values Dimensions“, in: *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing* (2014), S. 426–435.

gie ausgedrückt und unterstützt werden und wie sich ein Mangel an solcher Unterstützung negativ auszuwirken vermag.

All diese Ansätze benötigen Zeit und sind am produktivsten, wenn sie in einem frühen Stadium des Entwicklungsprozesses als Teil einer konzeptionellen Untersuchung von Werten und Schäden zum Einsatz kommen, und nicht erst als nachträgliche Fahndung nach Risiken.¹⁹¹ Die konzeptionellen Untersuchungen sollten durchgeführt werden, bevor sich die Forschenden auf bestimmte Ideen festlegen und daher weniger geneigt sind, ihren Kurs zu ändern, wenn sie mit Hinweisen auf mögliche Schäden konfrontiert werden. Dies bringt uns wieder zu dem Gedanken, mit dem wir diesen Abschnitt begonnen haben: dass die Forschung und Entwicklung von Sprachtechnologien, die sich gleichzeitig mit zutiefst menschlichen Daten befassen (nämlich Sprache) und Systeme schaffen, mit denen Menschen auf unmittelbare und lebendige Weise interagieren, mit äußerster Sorgfalt vorgehen sollten.

Abschließend wollen wir Anwendungsfälle von großen Sprachmodellen betrachten, die spezifisch marginalisierten Bevölkerungsgruppen von Nutzen sind. Wenn man, wie wir es befürworten, von der Entwicklung immer größerer Sprachmodelle absieht, muss man dann Vorteile aufgeben, die andernfalls solchen Gruppen zugutekommen würden? Nehmen wir als Beispiel die automatische Spracherkennung [*automatic speech recognition*, ASR], die dank großer Sprachmodelle einige Verbesserungen erfahren hat, sowohl in Bezug auf ihre Größe als auch auf die ihr zugrunde liegenden Architekturen¹⁹² (auch

¹⁹¹ Christopher A. Le Dantec, Erika Shehan Poole und Susan P. Wyche, „Values as Lived Experience: Evolving Value Sensitive Design in Support of Value Discovery“, in: *Proceedings of the SIGCHI conference on human factors in computing systems* (2009), S. 1141–1150.

¹⁹² Etwa Alexei Baevski und Abdelrahman Mohamed, „Effectiveness of Self-Supervised Pre-Training for ASR“, in: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing* (2020), S. 7694–7698; Chao-Wei Huang und Yun-Nung Chen, „Adapting Pretrained Transformer to Lattices for Spoken Language Understanding“, in: *Proceedings of 2019 IEEE Workshop on Automatic Speech Recognition and Understanding*

wenn die größten Sprachmodelle typischerweise zu umfangreich und zu langsam für die Anforderungen von Echtzeit-ASR-Systemen sind).¹⁹³ ASR hat viele positive Anwendungsbereiche, nicht zuletzt die automatische Untertitelung von Film- und Fernsehinhalten, was Gehörlosen und Schwerhörigen zugutekommt, die so Zugang zu ansonsten unzugänglichen Audioinhalten bekommen.¹⁹⁴ Wir sehen hier zwei Wege in die Zukunft: Der erste besteht in einer sehr viel breiter angelegten Suche nach Mitteln, ASR-Systeme zu verbessern – was im Übrigen bereits geschieht, da die Technologie nicht dazu geeignet ist, auf immer größere Sprachmodelle zurückzugreifen.¹⁹⁵ Aber selbst wenn es so wäre, wäre damit noch lange nicht der einzige effektive Weg zu einer besseren Spracherkennung bestimmt. (Wir weisen darauf hin, dass – wollen wir mächtige ASR für eine Vielzahl von Weltsprachen entwickeln – wir uns nicht darauf verlassen können, in allen Fällen Terabytes an Daten zu besitzen.) Sollten wir feststellen, dass große Sprachmodelle doch unumgänglich sind (und die notwendigen Daten zur Verfügung stehen), müssen wir diese Situation als Dual-Use-Problem behandeln und überlegen, wie wir die möglichen Schäden von Sprachmodellen, die als stochastische Papageien zur Anwendung kommen, auffangen und sie dennoch für die Verwendung in ASR-Systemen einsetzen können. Könnte man Sprachmodelle etwa so konstruieren, dass mit ihnen erzeugte synthetische Texte mit einem Wasserzeichen versehen und

(2019), S. 845–852; Joonbo Shin, Yoonhyung Lee und Kyomin Jung, „Effective Sentence Scoring Method Using BERT for Speech Recognition“, in: *Asian Conference on Machine Learning* (2019), S. 1081–1093.

¹⁹³ Hongzhao Huang und Fuchun Peng, „An Empirical Study of Efficient ASR Rescoring with Transformers“, in: *arXiv*, Oktober 2019, DOI: 10.48550/arXiv.1910.11450.

¹⁹⁴ Man bedenke aber, dass die automatische Untertitelung noch nicht gut genug ist – und wahrscheinlich nie gut genug sein wird –, um von Menschen erstellte Untertitel zu ersetzen. Außerdem bevorzugen Gehörlosengemeinschaften in manchen Kontexten menschliche Untertitel und eine Verdolmetschung in die entsprechende Gebärdensprache. Wir wollen in keinem Fall behaupten, dass automatische Systeme einen ausreichenden Ersatz für diese Schlüssel Anliegen der Barrierefreiheit darstellen.

¹⁹⁵ Huang und Peng, „ASR“.

somit identifizierbar wären?¹⁹⁶ Gibt es politische Ansätze, die ihre Verwendung wirksam regulieren können?

Zusammenfassend lässt sich sagen, dass wir für eine Forschung plädieren, die jene Menschen in den Mittelpunkt stellt, die von der entstehenden Technologie nachteilig betroffen sind, und die einen umfassenden Blick auf die möglichen Auswirkungen der Technologie auf Menschen behält. Das wiederum bedeutet, dass im Entwicklungsprozess Zeit für die genannten Faktoren bereitgestellt werden muss: für die Berücksichtigung von Umweltfolgen, eine sorgfältige Datenerfassung und -dokumentation, die frühzeitige Einbindung von Interessengruppen, die Erkundung alternativer Wege zur Erreichung langfristiger Ziele, die Beobachtung von Dual-Use-Szenarien und schließlich die Zuweisung von Forschungsmitteln zur Schadensbegrenzung, sollten solche Fälle eintreten.

8. Zusammenfassung

Seit die Verarbeitungsgeschwindigkeit von Computern zu den Möglichkeiten neuronaler Modelle aufgeschlossen hat, ist die Welt des NLP im Aufwind. Neuronale Ansätze im Allgemeinen und große Transformer-Sprachmodelle im Besonderen haben in einer Vielzahl von Benchmarks die bisherigen Spitzenreiter schnell überholt, und wieder einmal scheint sich die Binsenweisheit *there's no data like big data* zu bewahrheiten. Man könnte meinen, dass der gesamte Fortschritt in diesem Bereich tatsächlich von der Erstellung immer größerer Sprachmodelle

¹⁹⁶ Mikhail J. Atallah u.a., „Natural Language Watermarking and Tamperproofing“, in: *International Workshop on Information Hiding* (2002), S. 196–212; Nurul Shamimi Kamaruddin u.a., „A Review of Text Watermarking: Theory, Methods, and Applications“, in: *IEEE Access* 6 (2018), S. 8011–8028; Irene Solaiman u.a., „Release Strategies and the Social Impacts of Language Models“, in: *arXiv*, August 2019, DOI: 10.48550/arXiv.1908.09203.

(und der Erforschung ihrer Anwendungsmöglichkeiten) abhängt.

In diesem Paper haben wir unsere Leser:innen aufgefordert, einen Schritt zurückzutreten und sich zu fragen: Sind immer größere Sprachmodelle unvermeidlich oder gar notwendig? Welche Kosten sind mit dieser Forschungsrichtung verbunden und was sollten wir bedenken, bevor wir sie verfolgen? Braucht das Feld des NLP und die Öffentlichkeit, der es dient, wirklich immer größere Sprachmodelle? Wenn ja, wie können wir die damit verbundenen Risiken minimieren? Wenn nein, was ist stattdessen notwendig?

Wir haben eine Reihe von Kosten und Risiken aufgezählt, die mit dem Streben nach immer größeren Sprachmodellen verbunden sind. Dazu zählen *Umweltkosten* (die in der Regel von denjenigen getragen werden, die nicht von der daraus resultierenden Technologie profitieren); *finanzielle Kosten*, die wiederum Hürden errichten, durch die eingeschränkt wird, wer zu diesem Forschungsgebiet beitragen kann und welche Sprachen von den am weitesten fortgeschrittenen Techniken profitieren können; *Opportunitätskosten*, da Forscher:innen ihre Aufmerksamkeit von weniger ressourcenintensive Alternativen abziehen; und das *Schadensrisiko*, einschließlich Stereotypisierung, Verunglimpfung, Zunahme extremistischer Ideologie und unrechtmäßiger Verhaftung, wenn Menschen auf die scheinbar kohärenten Ausgaben von Sprachmodellen stoßen und diese für die Worte einer Person oder Organisation halten, die für das Gesagte zur Verantwortung gezogen werden kann.

Wir fordern daher die NLP-Forscherinnen und -Forscher auf, diese Risiken bei der Verfolgung dieser Forschungsrichtung sorgfältig abzuwägen und zu prüfen, ob die Vorteile die Risiken überwiegen. Sie sollten Dual-Use-Szenarien untersuchen, indem sie die vielen bereits existierenden Techniken nutzen, wie z.B. wertsensibles Design. Wir hoffen, dass diese Überlegungen die Forscherinnen und Forscher ermutigen,

Ressourcen und Anstrengungen in Technologien zu investieren, die NLP-Aufgaben effizient bearbeiten, ohne endlose Datenmengen zu verschlingen. Darüber hinaus rufen wir das Feld dazu auf, sich bewusst zu machen, dass Anwendungen, die glaubhaft versuchen, Menschen nachzuahmen, extreme Risiken in sich bergen. Eine klare rote Linie im Bereich der ethischen KI-Entwicklung ist die Arbeit an synthetischem menschlichen Verhalten. Hier gilt es, die Folgewirkungen zu verstehen und zu modellieren, um absehbaren Schaden für die Gesellschaft und einzelne gesellschaftliche Gruppen zu vermeiden. Wissenschaftliche Untersuchungen zu Nutzen, Nachteilen und Risiken der Imitation von Menschen sind daher ebenso erforderlich wie eine durchdachte Konzeption von Zielaufgaben, die auf hinreichend konkreten Anwendungsfällen basieren, um eine Zusammenarbeit mit den betroffenen Gemeinschaften zu ermöglichen.

Übersetzung Peter Meier

IMPRESSUM

Quellcodekritik. Zur Philologie von Algorithmen erscheint im August Verlag. Der August Verlag ist ein Forum für Theorie im Schnittpunkt von Philosophie, Politik und Kunst. Seit 2021 ist der August Verlag ein Imprint von Matthes & Seitz Berlin.

Dieses Buch ist als Open-Access-Publikation verfügbar über www.augustverlag.de und steht unter der Lizenz CC BY-NC-ND 4.0. (<http://creativecommons.org/licenses/by-nc-nd/4.0>).

August Verlag
august@augustverlag.de
www.augustverlag.de

Erste Auflage Berlin 2024

Copyright © 2024 Matthes & Seitz Berlin Verlagsgesellschaft mbH
Großbeerenstr. 57A, 10965 Berlin

Alle Rechte vorbehalten, insbesondere die Nutzung des Werkes für Text und Data Mining im Sinne von § 44b UrhG.

Gestaltung: Selitsch Weig
Druck: ArtDruk, Kobyłka

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind über <http://dnb.d-nb.de> abrufbar

Printed in Poland

ISBN 978-3-7518-9020-5

Diese Publikation wurde gefördert vom Schweizerischen Nationalfonds SNF.



Hannes Bajohr
SCHREIBENLASSEN
Texte zur Literatur im Digitalen

Format 11 x 18, Broschur
Auch als E-Book erhältlich
224 Seiten
Deutsch
1. Auflage 2022
ISBN 978-3-941360-97-6

Alle Literatur ist heute digital, aber nicht jede weiß darum. Die Frage, in welchem Sinne von digitaler Literatur gesprochen werden kann und was daraus für das Wissen über Literatur und Digitalität folgt, animiert die Beiträge dieses Bandes. In ihnen diskutiert Hannes Bajohr Verwandtschaftsverhältnisse zwischen konzeptueller und programmierter Literatur, skizziert Poetologien und Schreibpraxen und stellt sich der Herausforderung, die Künstliche Intelligenz sowie machine learning für das literarische Schreiben darstellen. So dokumentieren die zwischen 2014 und 2021 entstandenen Texte auch die Veränderungen in der Diskussion über Literatur im Digitalen. Sie erheben Einspruch gegen ein „prometheisches Unbehagen“, das die Ersetzung des Menschen durch die Maschine fürchtet und daher die Maschine nur menschlich denken kann. Welche Möglichkeiten ergeben sich stattdessen aus einer Literaturproduktion, die nicht mehr an einer anthropologischen Sonderstellung und Begriffen wie Genie oder Kreativität orientiert ist? Denn Literatur, so lässt sich hier erfahren, gibt es ohnehin nur als Verabredung.

„Als Praktiker, Wissenschaftler und kritischer Fürsprecher hat Hannes Bajohr mit diesem kurzweiligen Band eine lesenswerte Einführung geschrieben.“

Niels Penke, *Soziopolis*

„Besonders glänzt der Band einerseits dort, wo er die Diskursivität digitaler Literatur sowie der Digitalität im Allgemeinen in den Blick nimmt, andererseits dort, wo er digitale Literatur aus einer breiteren mediengeschichtlichen Perspektive betrachtet, Verbindungslinien und Brüche markiert, Gemeinsamkeiten und Unterschiede auslotet.“

Tobias Unterhuber, *Weimarer Beiträge*

„Originell und lesenswert“

Peter Krapp, *Monatshefte*

Louis Chude-Sokei
TECHNOLOGIE UND RACE
Essays der Migration

Mit einem Vorwort des Autors, aus dem Englischen von Utku Mogultay

Format 14 x 21, gebunden
Auch als E-Book erhältlich
272 Seiten
Deutsch
1. Auflage 2023
ISBN 978-3-7518-9011-3

Am Anfang der US-amerikanischen Unterhaltungskultur steht eine Schwarze Frau, von der das Publikum nicht weiß, ob es sich um einen 161 Jahre alten Menschen, einen Schwindel oder um einen Automaten handelt: Joice Heth, George Washingtons „Mammy“, mit der P. T. Barnum durch das Land tourt und schließlich die American Sideshow und den modernen Zirkus begründet. In Louis Chude-Sokeis eindringlichen Untersuchungen der Verschränkungen von Sklaverei, ihrem Nachleben und der technologischer Entwicklung ist die Geschichte der Sklavin, die bei der „Geburt Amerikas“ anwesend war, um ab 1835 zur nationalen Attraktion zu werden, einer der Ausgangspunkte für die Frage nach dem Menschsein im Zeichen der Technik. Ob im Minstrel oder in der Science-Fiction, in der Informationstheorie oder der Forschung zu Künstlicher Intelligenz: Stets erkennt er ein technisch-politisches Unbewusstes am Werk, das von Rassenängsten getrieben wird. Zugleich eröffnen Chude-Sokeis brillant geschriebene Essays den Raum für eine radikale Neubestimmung nicht nur der Wirklichkeit, sondern auch des Möglichen, indem sie die Verwandlung von und mit Technik im Kreolisierungsprozess, in der avantgardistischen Literatur und Philosophie der Karibik oder im Sound der Schwarzen Diaspora in den Blick nehmen.

„Wie können wir tatsächlich synthetische Körper erschaffen, ohne die Geschichte dessen zu reproduzieren, wie wir andere Körper in der Vergangenheit gewöhnlich behandelt, untersucht und eingeordnet haben?“

Louis Chude-Sokei

„Chude-Sokei betont, dass kulturelle Kreolisierung die Macht hat, die Entgegensetzung zwischen Mensch und ‚Untermensch‘, die das Korrelat der Sklaverei und des Kolonialismus bilden, zu überwinden und die Grundlage für ein neues Konzept des Menschseins zu bilden.“

Graham Douglas, *The Prisma*

Oliver Precht

DER ROTE FADEN

Maurice Merleau-Ponty und die Politik der Wahrnehmung

Enthält außerdem „L'Ineinander“ von Maurice Merleau-Ponty

Format 14 x 21, gebunden

288 Seiten

Deutsch

1. Auflage 2023

ISBN 978-3-7518-9016-8

Nicht nur in seinen heute weithin in Vergessenheit geratenen Texten zur Kommunistischen Partei, zu den Moskauer Prozessen, zum Lagersystem in der Sowjetunion, zur Situation der französischen Kolonien und zu vielen anderen Aspekten seiner Zeit, sondern durch sein gesamtes, ausuferndes Werk hindurch hat Maurice Merleau-Ponty eine filigrane ‚Politik der Wahrnehmung‘ entwickelt. Entlang zentraler Gedanken und Begriffe erklärt Oliver Precht die Entwicklung und die Vielstimmigkeit seines Werks und zeigt dessen unauflösliche Verflochtenheit in die Geschichte der Natur, der Politik und des Denkens auf. Das Buch nimmt diese grundlegende historisch-politische Situiertheit von Merleau-Pontys Denken zum Ausgangspunkt – und zeigt, dass es trotz oder vielmehr gerade aufgrund dieser Verflochtenheit in die eigene Zeit ein Angebot für die Gegenwart bereithält. Als umfassendes Porträt dieses ‚Denkers der Verflechtung‘ zeigt *Der rote Faden*, wie Merleau-Pontys radikal undogmatischer, existenzialistischer Marxismus die philosophischen Fundamente für eine linke Politik legt, die auch im Zeitalter des Anthropozäns nicht den Faden verliert.

Ergänzt wird das Buch durch einen hier erstmals übersetzten, fulminanten Text von Merleau-Ponty, der ursprünglich für sein unvollendet gebliebenes spätes Hauptwerk *Das Sichtbare und das Unsichtbare* vorgesehen war.

„Nun gibt es mit ‚Der rote Faden‘ eine großartige Einführung von Oliver Precht in das feinsinnige Denken Merleau-Pontys, das die frohe Botschaft des ‚L'Ineinander‘ verkündet.“

– Jakob Hayner, *Neues Deutschland*

McKenzie Wark

REVERSE COWGIRL

*Mit einem Nachwort der Autorin zur deutschen Ausgabe,
aus dem Englischen von Johanna Davids*

Format 11 x 18, gebunden
Auch als E-Book erhältlich
240 Seiten
Deutsch
1. Auflage 2023
ISBN 978-3-7518-9010-6

Was wäre, wenn du trans wärst und es nicht wüsstest? Wenn es eine Lücke in deinem Leben gäbe, von der du nicht mehr als eine unbestimmte Ahnung hast, dass es sie gibt? Wenn du dich nur im Drogenhoch oder beim Sex in deinem Körper zu Hause fühlst? Vom Sydney der 1980er Jahre bis ins heutige New York, in den sich wandelnden politischen und medialen Landschaften des späten zwanzigsten Jahrhunderts, spinnst *Reverse Cowgirl* eine Komödie der Irrungen. McKenzie Wark ist dabei, als aus dem Aufbruch von 1968, aus Punk, Disco und schwul-lesbischen Subkulturen neue Identitätsentwürfe entstehen – doch sie muss feststellen, dass sich ihr Leben weiterhin den Namen und Kategorien entzieht. Zwischen dem Versuch, als schwuler Mann zu leben, und jenem, als Mann mit Frauen zusammen zu sein, erkennt Wark, dass sie ganz anderer als der etablierten Erzählungen bedarf. Mit Anleihen bei den Genres der Autofiktion und Fiktionskritik entsteht so das gleichermaßen drastisch wie berührende Memoir einer Nichtexistenz: die Autoethnografie der Undurchsichtigkeit unseres Selbst.

„Eine Polybiografie, in der Wünsche darum kämpfen, jenseits der Grenzen des Patriarchats, des Kapitalismus, aber auch der schwulen Kultur zu existieren: ein Viertel Sperma und Arschficken, ein Viertel Punkglitzer und Schlampenpillen, ein Viertel kommunistische Theorie und du fügst den Rest hinzu.“

Paul B. Preciado

„Erkenntnisreich, lustvoll und manchmal auch sehr lustig liest sich der wilde Mix aus Pornographie und Existenzphilosophie, der sämtliche literarische Grenzen sprengt.“

Anja Kümmel, *Sissy Magazine*