GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Hausarbeit

im Studiengang "Digital Humanities & Allgemeine Sprachwissenschaften"

KI-Methoden im Akademischen Alltag

Emma Stein

Institut für Digital Huanities

31. August 2025

Georg-August-Universität Göttingen
Institut für Digital Humanities

Nikolausberger Weg 23
37073 Göttingen
Germany

☎ +49 (0)551 / 39-26791
✉ digitalhumanities@phil.uni-goettingen.de
https://www.uni-goettingen.de/de/597374.html

Erstbetreuer: Sascha Safenreider

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 31. August 2025

Inhaltsverzeichnis

1 Einführung

1.1 Künstliche Intelligenz im wissenschaftlichen Arbeiten

Künstliche Intelligenz wird heute in vielen Bereichen eingesetzt. Besonders im wissenschaftlichen Arbeiten sind zahlreiche KI-Tools entstanden, die bei verschiedenen Aufgaben helfen sollen [1, 2]

Die im Seminar behandelten Tools zeigten bereits, dass KI-Unterstützung Zeit und Ressourcen sparen kann. Da ich gerade meine Bachelorarbeit schreibe, bietet sich die Gelegenheit, verschiedene Tools praktisch zu testen. Die Bachelorarbeit eignet sich gut als Anwendungsbeispiel, da ich dadurch alle Bereiche des wissenschaftlichen Arbeitens abdecke, und ich die Tools sofort praktisch anwenden kann.

Dieser Bericht konzentriert sich daher auf die KI-Tools, die für meine Bachelorarbeit relevant sind. Das sind konkret Programmierung, Literaturrecherche und Schreiben.

1.2 Kontext der Bachelorarbeit

Für den Bericht ist es hilfreich ein bisschen Kontext über die Bachelorarbeit selbst zu haben: Meine Bachelorarbeit beschäftigt sich mit KI-Sicherheit, speziell mit dem Refusal-Verhalten von Sprachmodellen. Diese Modelle sind darauf trainiert, potentiell gefährliche Anfragen wie 'Wie baue ich eine Bombe?' oder 'Wie überfalle ich eine Bank?' abzulehnen [3]. Durch Methoden wie Adversarial Training [4] funktioniert dies bei englischen Anfragen meist gut, auch wenn es nach wie vor noch Jailbreak Möglichkeiten gibt [5].

Ein Problem zeigt sich jedoch wenn man andere Sprachen zum Prompten benutzt: Modelle antworten auf Spanisch, Deutsch oder Bengali häufiger auf unsichere Anfragen als auf Englisch, was teilweise an der unterschiedlichen Verfügbarkeit von Trainingsdaten liegt [6]. Eine Lösung durch Fine-tuning auf alle 7.000 Sprachen ist allerdings praktisch unmöglich aufgrund von den Zeit und Trainingsdaten, die dafür gebraucht werden und insbesondere in kleineren Sprachen nicht verfügbar sind [7]. Außerdem haben LLMs oft Probleme mit Language Confusion, wenn sie auf zu vielen Sprachen trainiert werden [8].

Mein Lösungsansatz verwendet Steering-Vektoren aus dem Bereich Representation Engineering, die das Refusal-Verhalten aus dem Englischen auf andere Sprachen übertragen sollen. Der Programmierpart ist zu 80% abgeschlossen, sodass jetzt die Schreibphase beginnt, für die noch etwa vier Wochen Zeit bleiben.

1.3 Struktur des Berichts

Der Bericht unterteilt sich in drei Bereiche: Programmierung, Literaturrecherche, und Schreiben. Für jeden Bereich beschreibe ich zunächst einige Metriken, anhand derer ich die Tools evaluiere. Anschließend werden die Tools selber kurz eingeführt und meine Nutzungserfahrung mit ihnen

erklärt. Am Ende von jedem dieser Subkapitel wird erneut Bezug auf die Metriken genommen und die Tools kurz anhand dieser verglichen.

2 Bericht

2.1 Programmierung

Da es in meiner Bachelorarbeit hauptsächlich darum geht, eine technische Lösung zu implementieren, programmiere ich viel. Beim Programmieren sind mir die folgenden Metriken wichtig:

- **Codequalität**. Wie gut ist der Code, den das Programm schreibt? Darunter fällt nicht nur, ob der Code funktioniert, sondern auch, ob er allgemeinen Best Practices folgt, und nicht zu komplex ist.
- **Geschwindigkeit**. Geschwindigkeit beim Programmieren bezeichnet, wie viel Zeit die KI Unterstützung spart.
- **Usability**. Wie angenehm ist das Tool zu nutzen?
- **Integrierung**. Wie nah ist das Tool an meine Codebase integrierbar?

2.1.1 Tools

Für das Programmieren werde ich die folgenden Tools ausprobieren: GitHub Copilot, Cursor sowie zwei LLMs (ChatGPT und Claude).

2.1.2 GitHub Copilot

GitHub Copilot[1] ist ein Programmierassistent, der von GitHub entwickelt wurde. Das Tool funktioniert als Plugin für verschiedene Entwicklungsumgebungen und schlägt Code-Vervollständigungen direkt im Editor vor. Copilot wurde auf öffentlichem Code trainiert und kann ganze Funktionen, Code-Blöcke oder einzelne Zeilen automatisch generieren[2]. Das Tool integriert sich nahtlos in den Programmierworkflow, da es kontextabhängige Vorschläge basierend auf dem bereits geschriebenen Code macht. Entwickler können die Vorschläge durch Drücken der Tab-Taste akzeptieren oder durch Weiterschreiben modifizieren. Copilot unterstützt eine Vielzahl von Programmiersprachen und kann sowohl bei der Implementierung neuer Funktionen als auch bei der Dokumentation von Code helfen.

GitHub Copilot nutze ich bereits seit einiger Zeit, da ich schon länger mit VS Code arbeite. Das Tool ist besonders hilfreich bei repetitiven Aufgaben, beispielsweise wenn Variablen konsistent in der gesamten Codebase geändert werden müssen. Die automatischen Vorschläge sparen in

[1] https://github.com/features/copilot
[2] https://github.com/features/copilot

diesen Fällen erheblich Zeit. Copilot ist gut darin, zum Beispiel aus Kommentaren entsprechende Codeblöcke einzubauen, und oft reicht es, Copilot durch Einfügen eines Kommentars indirekt zu prompten.

Oft trifft Copilot das, was gecodet werden soll akkurat genug, dass es nurnoch kleine oder sogar keine weiteren Modifikationen braucht. Was mich allerdings oft stört ist, dass man Copilot nicht sagen kann, wann Unterstützung gebraucht ist und wann nicht. Denn wenn ein Codevorschlag gemacht wird, man diesen aber nicht akzeptieren möchte, ist es oft schwierig, diesen Codeblock wieder verschwinden zu lassen, was ein kleines aber doch häufiges Ärgernis darstellt.

Copilot hat auch einen Agentenmodus: Dort kann man einen GPT4 basierten Agenten mit Copilot prompten ähnlich einem LLM. Der Agentenmodus hat mich allerdings bisher noch nicht überzeugt. Er hat tendenziell sehr lange Ladezeiten und die Vorschläge, die er liefert, sind zwar oft korrekt, aber nicht gut umgesetzt: Ein Beispiel war der Versuch, eine Funktion meiner Pipeline zu modifizieren, wobei Copilot die Änderungen in mein SLURM Job-Skript einfügen wollte, obwohl meine Pipeline auch im Kontext lag.

2.1.3 Cursor

Cursor[3] ist ein spezialisierter Code-Editor, der vom Aufbau her VS Code ähnelt, aber zusätzlich KI-Funktionalitäten direkt in die Entwicklungsumgebung integriert. Anders als reine Plugins bietet Cursor eine vollständige IDE-Erfahrung, die von Grund auf für die Zusammenarbeit mit KI konzipiert wurde. Das Tool ermöglicht es, Code durch natürliche Sprache zu generieren, zu bearbeiten und zu refaktorieren.

Ein besonderes Merkmal von Cursor ist die Möglichkeit, mit dem Code durch Chat-Interfaces zu interagieren, wobei der gesamte Projektkontext berücksichtigt wird[4]. Entwickler können spezifische Fragen an ihren Code stellen oder Änderungswünsche in natürlicher Sprache formulieren. Cursor unterstützt auch beim Debugging und kann Erklärungen für komplexe Code-Abschnitte liefern, was besonders bei der Arbeit mit fremdem oder älterem Code hilfreich ist.

Am Anfang war ich positiv überrascht von der Ähnlichkeit zu VS Code, was die Einarbeitung erheblich erleichterte. Nach Problemen zu Anfang mit häufigen Abstürzen funktioniert das Tool mittlerweile ziemlich zuverlässig.

Besonders angenehm ist die direkte Integration in die Codebase, die das mühsame Copy-Paste zwischen verschiedenen Anwendungen ersetzt. Die Code-Vorschläge sind qualitativ besser als die von Copilot, und die LLM-Sidebar mit Zugang zu großen Modellen wie Claude oder OpenAI übertrifft die entsprechende Funktion von Copilot deutlich. Das einzige nennenswerte Problem

[3] `https://docs.cursor.com/en/welcome`
[4] `https://docs.cursor.com/en/welcome`

sind gelegentliche Abstürze sowie die Tatsache, dass langfristige Nutzung kostenpflichtig wird, was allerdings für die meisten professionellen Tools in diesem Bereich gilt.

2.1.4 Large Language Models für die Programmierung

Generelle Large Language Models wie ChatGPT von OpenAI und Claude von Anthropic bieten umfassende Unterstützung beim Programmieren [9, 10]. Die Modelle können Code in verschiedenen Programmiersprachen generieren, debuggen und erklären. Der Vorteil dieser allgemeinen LLMs liegt in ihrer Vielseitigkeit und ihrer Fähigkeit, komplexe Zusammenhänge zu verstehen und zu erklären [11]. Entwickler können nicht nur nach Code-Lösungen fragen, sondern auch Diskussionen über verschiedene Implementierungsansätze führen. Allerdings erfordern diese Tools oft mehr manuelle Integration in den Workflow, da sie nicht direkt in die Entwicklungsumgebung eingebettet sind.

Allgemeine Large Language Models funktionieren für eine Vielzahl von Aufgaben sehr gut, neigen beim Programmieren jedoch dazu, unnötig komplexe Lösungen zu generieren als ich sie ausprobiert habe. Ein Beispiel war als ich nach einem Skript zum Einlesen von PyTorch .pt-Dateien gefragt habe, da VS Code diese nicht korrekt anzeigt. Anstatt der benötigten zehn Zeilen Code erhielt ich ein 243-zeiliges Konstrukt, das nicht funktioniert.

Störend ist auch, dass man den Code manuell zwischen der eigenen Codebase und dem LLM hin und her kopieren muss, besonders im Kontrast zu Cursor und Copilot. Dafür bieten diese LLMs jedoch maximale Flexibilität und können durch Abonnements auch für viele andere Aufgaben jenseits der Programmierung genutzt werden. Claude funktioniert meiner Erfahrung nach besser als ChatGPT, da es weniger umständliche Erklärungen liefert und über ein eigenes Code-Fenster verfügt, das verschiedene Versionen verwalten kann, ohne den gesamten Code neu generieren zu müssen.

Ich hatte nicht erwartet, dass ein Tool in allen vier Metriken für mich ganz oben stehen würde, aber Cursor ist nach meinem Experiment im Bezug auf Codequalität, Geschwindigkeit, Usability und insbesondere Integration definitiv am überzeugendsten. Danach steht Copilot und abschließend die großen LLMs, allerdings lassen sich diese Tools parallel zu Cursor etc. nutzen, da sie für verschiedene Zwecke genutzt werden können. Bei Copilot würde ich allerdings nicht den Agentenmodus nutzen, sondern Fragen, die ich dort gestellt hätte, an Claude weitergeben.

2.2 Literaturrecherche

Ein großer Teil wissenschaftlichem Arbeitens ist das Lesen und Integrieren der Papers anderer Forscher:innen in den eigenen Workflow. Momentan arbeite ich viel mit Obsidian[5] als einer Art Second Brain für die Literaturrecherche und bin damit sehr zufrieden. KI Unterstützung in diesem Bereich habe ich fast garnicht bisher benutzt, da ich bisher immer sehr skeptisch

[5] `https://obsidian.md`

gegenüber Halluzinationen bei der Recherche war. Allerdings bieten KIs hier großes Potenzial für Zeiteinsparungen, daher möchte ich sie gerne ausprobieren.

Für die Literaturrecherche sind mir folgende Kriterien wichtig:

- **Relevanz** der gefundenen Einträge. In der Vergangenheit habe ich öfter schon die Erfahrung gemacht, dass durch KI gefundene Einträge nicht wirklich zu meiner Query passen. Das lässt sich bestimmt durch Prompt-Engineering verbessern, allerdings müssen die gefundenen Paper einen Mindeststandard an Relevanz erreichen, damit ich sie verwenden kann.
- **Halluzinationsgrad**. Grade im Kontext wissenschaftlicher Arbeit sehr relevant, dass das LLM/KI-System sich keine Quellen ausdenkt oder Falschinformationen nennt [12] [13].
- **Usability**. Ein sehr praktischer Punkt ist die Usability des Systems: Wie einfach lassen sich gefundene Quellen exportieren, wie schnell kann ich sie gegenchecken und wie angenehm ist das Tool im Allgemeinen zu nutzen.

2.2.1 Tools

Die Tools, die mich im Bereich Literaturrecherche interessieren, sind Consensus AI, Elicit und ein Obsidian Plugin namens Copilot.

2.2.2 Consensus AI

Consensus AI [6] ist eine Suchmaschine spezialisiert auf wissenschaftliche Literatur. KI wird genutzt, um relevante Forschungsarbeiten zu finden und zusammenzufassen. Das Tool durchsucht große Datenbanken wissenschaftlicher Publikationen und bietet automatisch generierte Zusammenfassungen der wichtigsten Erkenntnisse zu spezifischen Forschungsfragen.

Ein besonderes Merkmal von Consensus ist die Möglichkeit, konsensbasierte Antworten zu liefern, indem es mehrere Studien zu einem Thema analysiert und die übereinstimmenden Ergebnisse hervorhebt[7]. Das Tool kann auch widersprüchliche Befunde identifizieren und dem Nutzer einen Überblick über den aktuellen Forschungsstand geben. Das ist besonders nützlich bei der Einordnung neuer Forschungsfragen in den bestehenden wissenschaftlichen Kontext.

Consensus AI liefert schon bei relativ kurzen Forschungsfragen gute und relevante Ergebnisse. Die Filtermöglichkeiten nach Journalqualität und die Option, Preprints auszuschließen, sind sehr nützlich für die gezielte Suche. Viele Tools erlauben es nicht, Preprints auszuschließen, aber für das wissenschaftliche Arbeiten sind Peer-reviewed Papers immernoch der Goldstandard.

Die Antwortzeiten auf einen Prompt sind erfreulich kurz. Ein Problem zeigt sich jedoch bei der Vorschlagqualität, da das Tool sehr häufig Arbeiten mit geringen Zitationszahlen empfiehlt, was allerdings auch themenspezifisch bedingt sein könnte. Das Thema, das ich in meiner Bachelorarbeit

[6] https://consensus.app
[7] https://consensus.app

behandle, ist relativ neu. Problematischer sind die oft fehlerhaften Metainformationen: Links zu Papern fehlen regelmäßig und die angegebenen Zitationszahlen sind teilweise inkorrekt. Positiv ist jedoch, dass bisher keine echten Halluzinationen aufgetreten sind und die fehlerhaften Informationen wie eine fehlende DOI schnell zu korregieren sind.

2.2.3 Elicit

Elicit[8] ist ein KI-Forschungsassistent, der speziell für die Unterstützung bei Literaturreviews und Meta-Analysen entwickelt wurde. Das Tool kann automatisch relevante Paper identifizieren, die wichtige Informationen extrahieren und strukturierte Übersichten erstellen.

Das Tool ermöglicht es, spezifische Forschungsfragen zu stellen und man erhält systematische Antworten basierend auf der verfügbaren Literatur [9]. Elicit kann auch dabei helfen, Forschungslücken zu identifizieren und neue Hypothesen zu generieren. Besonders wertvoll ist die Fähigkeit des Tools, Daten aus verschiedenen Studien zu extrahieren und in tabellarischer Form zu präsentieren, was die Vergleichbarkeit von Forschungsergebnissen erheblich erleichtert.

Elicit braucht wesentlich mehr Zeit für die Bearbeitung von Anfragen als Consensus, bietet dafür aber deutlich detailliertere Ergebnisse. Besonders schätze ich die Gestaltung der Eingabemaske, die Nutzer dazu bringt, präzisere Prompts zu formulieren, was die Qualität der Antworten erheblich verbessert. Zum Beispiel zeigt es an, dass man Metriken oder mehr Informationen zum Prompt hinzufügen soll.

Die Möglichkeit, Quellen direkt im Editor einzusehen, spart erheblich Zeit und reduziert das Navigieren zwischen verschiedenen Tabs. Was ich auch nennenswert fand, war die Fähigkeit des Tools, wichtige Parameter automatisch zu erkennen - bei meiner Anfrage zu Safety Alignment-Methoden in verschiedenen Sprachen hat es die untersuchten Sprachen, verwendeten Methoden und methodologischen Ansätze jeder Studie erklärt. Tatsächlich sind die genauen Sprachen sehr wichtig für meine Forschung und ich hätte nicht erwartet, dass das Tool das so schnell erkennt.

Die Möglichkeit, Dinge direkt als BibTeX-Datei zu exportieren, erleichtert das Literaturmanagement außerdem erheblich beim Schreiben. Das kostenlose Kontingent ist jedoch schnell erschöpft, was bei vermehrter Nutzung kostenpflichtige Abonnements erfordert.

2.2.4 Copilot Plugin für Obsidian

Das letzte Tool ist ein bisschen weniger bekannt, aber ganz interessant für mich, da ich in Obsidian[10] quasi alle meine Papers liegen habe. Das Plugin[11] integriert KI-Funktionalitäten direkt in Obsidian und verwandelt es so in einen intelligenten Forschungsassistenten. Das Plugin bietet verschiedene

[8] `https://elicit.com`
[9] `https://elicit.com`
[10] `https://obsidian.md`
[11] `https://www.obsidiancopilot.com/en`

Funktionen wie chat-basierte Suche im eigenen Notizarchiv, Websuche und die Verarbeitung verschiedener Medienformate wie PDFs, Videos und Webseiten.

Ein zentrales Feature ist der SSmart Vault Search", der es ermöglicht, durch eigene Prompts in den eigenen Notizen zu suchen, ohne dass eine aufwendige Indexierung erforderlich ist. Das Plugin unterstützt auch einen "Project Mode", der Kontext basierend auf Ordnern und Tags erstellt, ähnlich z.B. NotebookLM. Für Premium-Nutzer gibt es außerdem einen autonomen Agent-Modus, der automatisch relevante Tools und Suchfunktionen aktiviert, ohne dass spezifische Befehle eingegeben werden müssen. Das Plugin kann auch verschiedene AI-Modelle verwenden und bietet vollständige Kontrolle über die eigenen Daten dadurch dass Daten lokal gespeichert werden.

Das Copilot Plugin ist besonders interessant, da ich meine gesamte Wissensdatenbank und Papers in Obsidian verwalte. Das Setup verlief überraschend reibungslos, was bei Community-Plugins nicht immer gewährleistet ist. Die angebotenen Funktionalitäten sind technisch gut umgesetzt, der praktische Mehrwert blieb für mich jedoch begrenzt. Da ich bereits das Backlink-System von Obsidian nutze und meine Bibliothek hauptsächlich graphisch über vernetzte Notizen erkunde, brachten die KI-Features keine erhabliche Verbesserung meines Workflows. Viele Funktionen wie das "Temporal Understandingßcheinen eher für persönliche Notizen als für wissenschaftliche Recherche konzipiert zu sein. Es ist gut möglich, dass andere Nutzer größeren Nutzen aus dem Plugin ziehen können, aber für mich hatte das Tool in Obsidian keinen großen Mehrwert.

Ganz allgemein war ich bei der Erprobung der Literaturrecherche-Tools aber sehr positiv überrascht von der Relevanz der gefundenen Literatur. Vor diesem Test hatte ich KI-gestützte Literaturrecherche grundsätzlich abgelehnt, auf Basis von früheren Erfahrungen mit allgemeinen LLMs wie ChatGPT, die zwar Literatur finden, aber oft halluzinieren oder nicht relevante Papers finden.

Der entscheidende Unterschied ist anscheinend, spezialisierte KI-Tools zu nutzen, die gezielt auf das Finden wissenschaftlicher Literatur trainiert wurden. Bei keinem der getesteten Tools gab es große Probleme mit Halluzinationen, was ein großer Fortschritt gegenüber allgemeinen Sprachmodellen darstellt. Eine bessere Integration in wissenschaftliche Tools wie Overleaf oder Zotero, würde die Tools noch besser machen, aber dennoch sind die Tools auch so bereits eine gute Ergänzung.

2.3 Schreiben und Editieren

Beim Schreiben gibt es nur ein Tool, das ich ausprobieren möchte. Es ist eher experimentell, aber dafür sehr interessant: Es handelt sich um ein GitHub-Repository[12], das LLMs, Zotero und LaTeX verbindet. Dieses Repository wurde von einem Freund von mir geschrieben, er schreibt damit regelmäßig Paper und ich wollte es schon länger ausprobieren. Die Metriken, die ich mir dafür anschaue sind:

[12] https://github.com/wusche1/cursor-latex-zotero-template

- **Textqualität**. Wie natürlich klingt der Text? Hat der Text Fehler?
- **Integrierung**. Wie gut ist das Tool im Schreibprozess zu benutzen? Lässt es sich an andere akademische Tools wie Zotero oder Overleaf anknüpfen?
- **Anpassbarkeit** Wie sehr lässt sich das Tool an meine eigenen Bedürfnisse anpassen?

Das Repository löst einige häufige Probleme beim Arbeiten mit LLMs und Literatur. Zum Beispiel extrahiert es automatisch aus dem PDF eines Papers in Zotero eine Markdown Datei, da LLMs meiner Erfahrung nach oft nicht gut mit PDFs arbeiten können. Man kann seine eigenen Notizen in einer Markdown Datei schreiben und anschließend generiert das LLM ausgehend von diesen Notizen ausgeschriebenen Text in ein LateX Template, das man selber verändern kann.

Ein positiver Punkt ist, dass das Setup sehr flexibel ist: Die Regeln im Prompt der LLMs können selber angepasst werden, auch können eigene LateX Templates genutzt werden. Ganz allgemein ist dieses Tool komplett Open-Source auf GitHub verfügbar, was es sehr angenehm zum Anpassen macht und Kosten spart im Vergleich zu anderen Integrationen.

Dass Notizen und Generierung getrennt stattfindet, fand ich sehr praktisch, da man an den Notizen Dinge ändern kann, ohne direkt den kompletten Text ändern zu müssen.

Durch die eingebauten Zotero Files wirkt das Tool wie ein RAG-System für die eigene Literatur, ohne zu viel Komplexität.

Ein Nachteil ist sicherlich, dass das Repo nicht professionell gewartet oder gedebuggt wird, also muss man mehr Zeit investieren, bis das Tool funktioniert. Zum Beispiel kann das Repository im Moment nur auf Windows oder MacOS laufen, nicht aber auf Linux.

Außerdem wurde eine Funktion angepriesen als:

> **Fact-checking and verification**: AI can verify claims against the actual paper content, suggest relevant quotes, and ensure citation accuracy[13]

. Eigentlich ist diese Funktion allerdings nur ein einziger Call an ein LLM, der fragt, ob das LLM nachschauen kann, ob die Information wirklich existiert, was für mich nicht genügend ist als Qualitätsmaß. Die Idee eines solchen "Faktenchecksïntegriert in ein Tool ist sehr gut, nur bringt ein solcher Check nichts, wenn man sich nicht auf ihn verlassen kann, da man dann doch die Zitation nachschauen muss.

Die Integration dieses Tools ist sicherlich sehr gut, wesentlich besser als die "klassischen"LLMs, bei denen man hin- und herwechseln muss zwischen geschriebenem Text und Chatwindow. Die Textqualität ist dabei ungefähr gleich gut wie bei den großen LLMs, da das integrierte LLM diese LLMs nutzt. Allerdings machen die integrierten Regeln es einfacher den Stil des LLMs anzupassen und festzuhalten, sobald man gute Prompts gefunden hat.

[13] https://github.com/wusche1/cursor-latex-zotero-template/blob/main/README.md

3 Fazit

Als jemand, der bisher im Bereich KI Unterstützung eigentlich nur LLMs genutzt hat, war es sehr interessant, eine größere Anzahl an Tools für meine Bachelorarbeit zu verwenden, da viele davon sehr nützlich sind.

Natürlich benötigen diese Tools mehr Tests: Die meisten Tools wurden einen Tag lang getestet, aber die besten Tools werde ich in meinen Workflow einbauen. Insbesondere Cursor und die Tools zur Literaturrecherche sind Software, die ich gerne mehr einbauen möchte. Mit meiner Bachelorarbeit in Arbeit werden sich dafür auch gute Gelegenheiten bieten.

Was ich allerdings auch gemerkt habe bei der Arbeit mit diesen Tools ist, dass man unbedingt auf die Tendenz achten muss, sich zu viel auf KI Unterstützung zu verlassen Grade wenn die Anreize so gesetzt sind, dass man entweder nochmal 20 Minuten mehr Literaturrecherche machen könnte oder einfach die Papers, die das LLM vorschlägt, annimmmt, gibt es die Gefahr, dass die Recherche am ende nicht vollständig ist [14].

Außerdem mache ich mir grade in der Wissenschaft sorgen, wo die Anzahl an Zitationen eines Papers eine wichtige Metrik ist [15], dass KI gestützte Systeme diese Nummern verzerren weil sie zum Beispiel nur Papers aus bestimmten Datenbanken oder Keywords empfehlen.

Was für mich weiterhin ein wichtiger Grundsatz bei der Arbeit mit KI Tools ist, ist dass man KI Tools benutzen kann, sobald man sich gut genug mit einem Gebiet auskennt, um Fehler zu erkennen. Bei Coding KIs zum Beispiel kann KI für Funktionen sehr hilfreich sein, wenn man die oft notwendigen kleinen Anpassungen vornehmen kann und die Funktion ordentlich testen kann. Wenn man sich aber auf einem Gebiet nicht auskennt und KI Unterstützung nutzt, hat es großes Fehlerpotential, weil man Nutzen nicht von Halluzination unterscheiden kann.

Alles in Allem ist KI Unterstützung aber für das wissenschaftliche Arbeiten eine große Unterstüzung, die viel Zeit sparen kann, wenn man sie in Kombination mit angemessener Vorsicht und eigenem Wissen nutzt.

Literatur

[1] H. Wang, T. Fu, Y. Du, W. Gao, K. Huang, Z. Liu, P. Chandak, S. Liu, P. Van Katwyk, A. Deac, A. Anandkumar, K. Bergen, C. P. Gomes, S. Ho, P. Kohli, J. Lasenby, J. Leskovec, T.-Y. Liu, A. Manrai, D. Marks, B. Ramsundar, L. Song, J. Sun, J. Tang, P. Veličković, M. Welling, L. Zhang, C. W. Coley, Y. Bengio, and M. Zitnik, "Scientific discovery in the age of artificial intelligence," *Nature*, vol. 620, no. 7972, pp. 47–60, Aug. 2023. [Online]. Available: https://www.nature.com/articles/s41586-023-06221-2

[2] A. A. Oyelude, "Artificial intelligence (AI) tools for academic research," *Library Hi Tech News*, vol. 41, no. 8, pp. 18–20, Oct. 2024. [Online]. Available: http://www.emerald.com/lhtn/article/41/8/18-20/1233085

[3] N. Varshney, P. Dolin, A. Seth, and C. Baral, "The Art of Defending: A Systematic Evaluation and Analysis of LLM Defense Strategies on Safety and Over-Defensiveness," 2024. [Online]. Available: https://arxiv.org/abs/2401.00287

[4] A. Paulus, A. Zharmagambetov, C. Guo, B. Amos, and Y. Tian, "AdvPrompter: Fast Adaptive Adversarial Prompting for LLMs," 2024. [Online]. Available: https://arxiv.org/abs/2404.16873

[5] X. Li, R. Wang, M. Cheng, T. Zhou, and C.-J. Hsieh, "DrAttack: Prompt Decomposition and Reconstruction Makes Powerful LLM Jailbreakers," 2024. [Online]. Available: https://arxiv.org/abs/2402.16914

[6] W. Wang, Z. Tu, C. Chen, Y. Yuan, J.-t. Huang, W. Jiao, and M. R. Lyu, "All Languages Matter: On the Multilingual Safety of Large Language Models," 2023. [Online]. Available: https://arxiv.org/abs/2310.00905

[7] Y. Xu, L. Hu, J. Zhao, Z. Qiu, K. Xu, Y. Ye, and H. Gu, "A survey on multilingual large language models: Corpora, alignment, and bias," *Frontiers of Computer Science*, vol. 19, no. 11, p. 1911362, Nov. 2025. [Online]. Available: https://link.springer.com/10.1007/s11704-024-40579-4

[8] K. Marchisio, W.-Y. Ko, A. Berard, T. Dehaze, and S. Ruder, "Understanding and Mitigating Language Confusion in LLMs," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Miami, Florida, USA: Association for Computational Linguistics, 2024, pp. 6653–6677. [Online]. Available: https://aclanthology.org/2024.emnlp-main.380

[9] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an LLM to Help With Code Understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. Lisbon Portugal: ACM, Apr. 2024, pp. 1–13. [Online]. Available: https://dl.acm.org/doi/10.1145/3597503.3639187

[10] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-Collaboration Code Generation via ChatGPT," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 7, pp. 1–38, Sep. 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3672459

[11] Z. Zheng, K. Ning, Q. Zhong, J. Chen, W. Chen, L. Guo, W. Wang, and Y. Wang, "Towards an Understanding of Large Language Models in Software Engineering Tasks," 2023. [Online]. Available: https://arxiv.org/abs/2308.11396

[12] F. Liu, Y. Liu, L. Shi, H. Huang, R. Wang, Z. Yang, L. Zhang, Z. Li, and Y. Ma, "Exploring and Evaluating Hallucinations in LLM-Powered Code Generation," 2024. [Online]. Available: https://arxiv.org/abs/2404.00971

[13] M. Chelli, J. Descamps, V. Lavoué, C. Trojani, M. Azar, M. Deckert, J.-L. Raynier, G. Clowez, P. Boileau, and C. Ruetsch-Chelli, "Hallucination Rates and Reference Accuracy of ChatGPT and Bard for Systematic Reviews: Comparative Analysis," *Journal of Medical Internet Research*, vol. 26, p. e53164, May 2024. [Online]. Available: https://www.jmir.org/2024/1/e53164

[14] J. G. Meyer, R. J. Urbanowicz, P. C. N. Martin, K. O'Connor, R. Li, P.-C. Peng, T. J. Bright, N. Tatonetti, K. J. Won, G. Gonzalez-Hernandez, and J. H. Moore, "ChatGPT and large language models in academia: Opportunities and challenges," *BioData Mining*, vol. 16, no. 1, pp. 20, s13 040–023–00 339–9, Jul. 2023. [Online]. Available: https://biodatamining.biomedcentral.com/articles/10.1186/s13040-023-00339-9

[15] D. W. Aksnes, L. Langfeldt, and P. Wouters, "Citations, Citation Indicators, and Research Quality: An Overview of Basic Concepts and Theories," *Sage Open*, vol. 9, no. 1, p. 2158244019829575, Jan. 2019. [Online]. Available: https://journals.sagepub.com/doi/10.1177/2158244019829575

[16] J. Wahle, T. Ruas, S. M. Mohammad, N. Meuschke, and B. Gipp, "Ai usage cards: Responsibly reporting ai-generated content," in *2023 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2023, pp. 282–284. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/JCDL57899.2023.00060

AI Usage Card

PROJECT DETAILS	PROJECT NAME KI im wissenschaftlichen Arbeiten	DOMAIN	KEY APPLICATION Editieren
CONTACT(S)	NAME(S)	EMAIL(S)	AFFILIATION(S)
MODEL(S)	MODEL NAME(S) Claude-4-Sonnet, Consensus		
LITERATURE REVIEW	FINDING LITERATURE Consensus wurde benutzt, um Quellen zu finden für die Einleitung. Die Quelle im Bezug auf meine Bachelorarbeit sind nicht mithilfe von KI Assistenz gefunden.	FINDING EXAMPLES FROM KNOWN LITERATURE OR ADDING LITERATURE FOR EXISTING STATEMENTS -	COMPARING LITERATURE -
WRITING	GENERATING NEW TEXT BASED ON INSTRUCTIONS -	ASSISTING IN IMPROVING OWN CONTENT OR PARAPHRASING RELATED WORK Claude-4-Sonnet wurde genutzt, um bereits geschriebenen Text zu verbessern. Es wurde keine KI genutzt, um Text von Grund auf zu schreiben.	PUTTING OTHER WORKS IN PERSPECTIVE -
CODING	GENERATING NEW CODE BASED ON DESCRIPTIONS OR EXISTING CODE	REFACTORING AND OPTIMIZING EXISTING CODE	COMPARING ASPECTS OF EXISTING CODE

ETHICS	WHY DID WE USE AI FOR THIS PROJECT?	WHAT STEPS ARE WE TAKING TO MITIGATE ERRORS OF AI?	WHAT STEPS ARE WE TAKING TO MINIMIZE THE CHANCE OF HARM OR INAPPROPRIATE USE OF AI?
		Quelle wurden geprüft nach Vorschlag durch Consensus	

THE CORRESPONDING AUTHORS VERIFY AND AGREE WITH THE MODIFICATIONS OR GENERATIONS OF THEIR USED AI-GENERATED CONTENT

AI Usage Card v2.0 https://ai-cards.org [16]