Lessons Learned 5, 2 (2025)
Submitted: 22.09.2025
Accepted: 08.01.2026

DOI: https://doi.org/10.25369/ll.v5i2.105
ISSN: 2749-1307

Programmieren für die Geistes- und Sozialwissenschaften

Y. Frommherz*, S. Meier-Vieracker, A. J. Matz

Professur für Angewandte Linguistik, Institut für Germanistik und Medienkulturen, Fakultät Sprach-, Literatur- und Kulturwissenschaften, TU Dresden

Abstract

Die fortschreitende Digitalisierung in den Geistes- und Sozialwissenschaften (*GSW*) ermöglicht und erfordert den Einsatz neuer, insbesondere quantitativer Analysemethoden. Den flexibelsten Umgang mit Daten erhalten Forschende durch Programmierkenntnisse. Typischerweise werden diese aber nicht in GSW-Studiengängen vermittelt und bestehende Lehr-/ Lernmaterialien setzen meist technisches Vorwissen voraus. Deshalb hat das *virTUos*-Teilprojekt *ExDiMed* niedrigschwellige, modular aufgebaute Selbstlernmaterialien zur Vermittlung von Codeskills in der Programmiersprache Python entwickelt, die sich spezifisch an den Bedarfen und Voraussetzungen GSW-Studierender orientieren. Die Inhalte in Form interaktiver Jupyter Notebooks stehen Lernenden wie Lehrenden als OER zur Verfügung und wurden an der TU Dresden während sechs Semestern in einem Flipped Classroom-Seminar erfolgreich erprobt und verfeinert. Mit dem Aufkommen generativer KI wurde das Angebot explorativ erweitert, um Studierende für konstruktiven KI-Einsatz beim Coden zu sensibilisieren. Unsere Evaluation zeigt, dass die Vermittlung grundlegender Programmierkenntnisse in Zeiten von KI an Relevanz gewonnen hat, da letztere nur dann als Katalysator der Codepraxis wirken kann, wenn Studierende bereits über Basics verfügen. Unser Lehr-/Lernkonzept ist mittlerweile im Masterstudiengang *Digital Humanities* an der TU Dresden verankert und wird auch nach Projektende weiterentwickelt.

The ongoing digitization in the humanities and social sciences (*HSS*) enables and requires the use of new, particularly quantitative, analysis methods. Programming skills give researchers the most flexible way to handle data. However, these are not typically taught in HSS degree programs, and existing teaching/learning materials usually require prior technical knowledge. For this reason, the *virTUos* subproject *ExDiMed* has developed low-threshold, modular self-learning materials for teaching coding skills in the programming language Python, which are specifically tailored to the needs and requirements of HSS students. The content, in the form of interactive Jupyter Notebooks, is available to learners and teachers as OER and has been successfully tested and refined at TU Dresden over six semesters in a flipped classroom seminar. With the advent of generative AI, the course has been expanded in an exploratory manner to sensitize students to the constructive use of AI in coding. Our evaluation shows that teaching basic programming skills has become more relevant in the age of AI, as the latter can only act as a catalyst for coding practice if students already have the basics. Our teaching/learning concept is now firmly established in the Master's program in *Digital Humanities* at TU Dresden and will continue to be developed even after the project ends.

*Corresponding author: yannick.frommherz@tu-dresden.de

1. Einleitung

Die Geistes- und Sozialwissenschaften (GSW) haben in den letzten Jahren einen digital turn erfahren [1], [2], [3], [4], der durch jüngste Entwicklungssprünge generativer KI zusätzlich beschleunigt werden dürfte [5], [6], [7]. Mittlerweile stehen Wissenschaftler:innen dieser Disziplinen große Mengen an digitalen Daten zur Verfügung (z. B. Social Media-Korpora oder OCR-Digitalisate), die den Einsatz quantitativer Analysemethoden gleichermaßen ermöglichen und erfordern. Neben niedrigschwelligen Tools, die bestimmte, vordefinierte Auswertungen erlauben (z. B. Korpusanalysesoftware wie *SketchEngine*), versetzen erst Programmierkenntnisse GSW-Forschende in die Lage, große Quantitäten an für sie zielführenden Daten flexibel zu erschließen und zu bereinigen, auszuwerten sowie für die Dissemination aufzubereiten [8].

Angesichts des traditionell eher qualitativen Fokus vieler GSW sind quantitative Methoden im Allgemeinen und Programmierskills im Besonderen aber typischerweise nicht Bestandteil entsprechender Curricula. Vor diesem Hintergrund setzte sich das Teilprojekt *Experimentierraum Digitale Medienkompetenz* (*ExDiMed*) des *virTUos*-Verbunds 2021 zum Ziel, ein adressatengerechtes Lehr-/Lernformat zu entwickeln, das genau diese Kompetenzen an GSW-Studierende vermittelt, um ihnen so zu ermöglichen, das Potenzial des digital turn voll auszunutzen [8].

Konkret haben wir während der Projektlaufzeit Lehr-/Lernmaterialien erarbeitet und erprobt, die Programmierskills in Python als fundamentale digitale Medienkompetenz vermitteln. Denn als de-facto-Wissenschaftsstandard bietet diese relativ einfach zu erlernende Programmiersprache die Möglichkeit, Forschungsfragen und Anwendungsprobleme aus den GSW bedarfsgerecht zu lösen [8], [9].

Dieser Beitrag reflektiert die Arbeit im Rahmen von *ExDiMed*, beginnend mit den spezifischen Bedarfen und Voraussetzungen von GSW-Studierenden, gefolgt von unseren darauf aufbauenden Selbstlernmaterialien, deren Erprobung in einem Flipped Classroom-Seminar sowie der sich in der Zwischenzeit aufdrängenden Integration generativer KI. Wir schließen mit einem Ausblick in die Zukunft. Während des gesamten Projekts flossen iterativ gesammelte Rückmeldungen seitens der Lernenden zu unserem Ansatz sowie den konkreten Materialien in deren Weiterentwicklung ein (s. Schema des Arbeitsprozesses in Abb. 1). Ganz dem Titel dieser Zeitschrift entsprechend wollen wir die wichtigsten *Lessons Learned* aus unserem Projekt präsentieren.

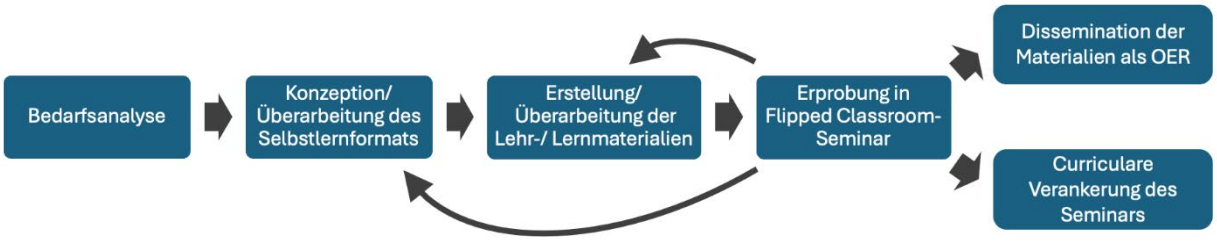


Abb. 1: Schematische Darstellung des iterativen Arbeitsprozesses.

2. Ausgangslage

Hochqualitative Einführungen ins Programmieren, die außerdem frei im Internet verfügbar sind, liegen vielfach und in verschiedenen Formaten vor, von klassischen lehrbuchähnlichen Websites über YouTube-Tutorials hin zu interaktiven Online-Kursen. Abgesehen von nennenswerten Ausnahmen wie *Programming Historian* werden darüber aber fast ausschließlich Lernende mit einer gewissen technischen, mathematischen und logischen Grundaffinität adressiert. So setzen Erklärungen häufig einschlägiges Wissen voraus und Beispiele sowie Übungen sind zumeist numerischer Natur (z. B. die Ausgabe der Fibonacci-Folge). Unsere Zielgruppe, Studierende der GSW, verfügt aber in aller Regel über keinerlei Vorkenntnisse im Programmieren sowie generell über wenig technisches Know-how. Im Gegenteil, häufig

herrschen gar Vorbehalte gegenüber computationeller Arbeit vor. Abgesehen von der Wichtigkeit möglichst voraussetzungsfreier Inhalte (s. u.) stand deshalb bei der Entwicklung unseres Lehr-/Lernangebots zunächst die Frage im Zentrum, über welches Format Programmierkenntnisse am zielführendsten an das gegebene Publikum vermittelt werden können.

3. Open-Source Selbstlernmaterialien

Während unsere Zielgruppe vereint, dass sie technisch oft wenig erfahren ist und Textdaten für sie i. d. R. einen hohen Stellenwert haben, sind die unterschiedlichen GSW doch thematisch und methodisch breitgefächert. So bringen etwa Studierende der Geschichtswissenschaften andere Konzepte und Interessenslagen mit als jene der Linguistik. Über die Fächer hinaus haben zudem alle Studierenden je individuelle Bedarfe und Voraussetzungen. Um diese Vielfalt zu berücksichtigen und ein möglichst breites Publikum innerhalb der GSW zu adressieren, haben wir uns für *Selbstlernmaterialien* im Format der Jupyter Notebooks entschieden, die außerdem *modular* aufgebaut sind.

Sobald Studierende unsere frei bei GitHub zugänglichen Inhalte heruntergeladen haben, können sie diese flexibel selbst bearbeiten, wann und wo sie wollen und – ganz entscheidend – in ihrem eigenen Tempo. Dadurch können auch individuell herausfordernde Passagen repetiert werden, so oft wie nötig. Vorab von Testpersonen sowie im Anschluss an die Seminare eingeholtes Feedback hat uns darin bestärkt, dass Lernende diese Flexibilität sehr schätzen.

Jupyter Notebooks eignen sich als Format, da sie auf elegante, technisch unaufdringliche Weise Code- mit Textblöcken verweben (s. Abb. 2). Letztere beinhalten Erklärungen sowie Aufgabenstellungen und sind für Lernende nicht editierbar. Codefelder wiederum sind entweder leer (z. B. bei Übungen, s. Abb. 3) oder enthalten bereits Code, der von den Studierenden erweitert („Schreibe den Code fertig“), modifiziert („Setze relevante Parameter ein“ bzw. „Finde den Fehler“) und in jedem Fall ausgeführt werden kann. Im Gegensatz zu klassischen Lehrbüchern, die mühsames Abtippen des Codes erfordern, aber auch skriptbasierten Tutorials, die erst in der Command Line ausgeführt werden müssen, erfahren Lernende in unseren interaktiven Notebooks unmittelbar, was der gegebene Code bzw. bestimmte Änderungen daran bewirken. Die Unterteilung von Code in mehrere kleine Blöcke (ggf. mit Text dazwischen) sensibilisiert weiter für den inkrementellen Charakter von Programmieren: Schritt für Schritt wird ein Problem gelöst und nach jedem Block wird direkt sichtbar, ob der Code das gewünschte (Zwischen-)Resultat liefert. Unser Lehr-/Lernformat kombiniert also selbstgesteuerte Wissensaneignung mit unmittelbarer Anwendung. Eingebettete Grafiken und Videos, die etwa ein realistisches Bild von explorativem Trial-and-Error-Coding vermitteln, schaffen zudem ein multimodales Lernerlebnis.

Abb. 2: Ausschnitt aus einem Jupyter Notebook bestehend aus Text- und Codeblöcken, die in bedingte Anweisungen einführen. Die Codeblöcke sind ausgeführt.

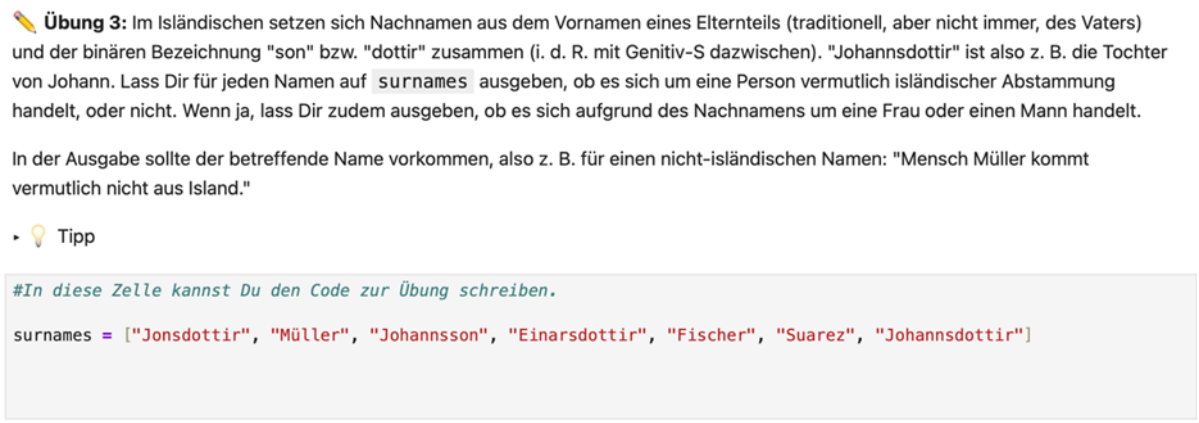


Abb. 3: Beispiel einer Übung zu bedingten Anweisungen, die thematisch den GSW entlehnt ist. Optional kann ein Tipp eingeblendet werden, der eine geeignete Methode (`endswith`) vorschlägt.

Inhaltlich sind unsere Notebooks wie erwähnt *modular* aufgebaut. Wir unterscheiden zwischen einem Grundlagenmodul und Aufbaumodulen. In ersterem eignen sich Studierende das theoretische und praktische Fundament beim Programmieren mit Python an, d. h. Kenntnisse zu Variablen, Operatoren, Datentypen, Kontrollstrukturen, nützlichen Funktionen und Methoden sowie dem Umgang mit externen Daten. Mit Blick auf unsere Zielgruppe sind Erklärungen, Beispiele und Übungen dabei bewusst den GSW entnommen und praxisnah ausgelegt (z. B. wird die Verteilung der Redebeiträge in *Nathan der Weise* analysiert, s. auch Abb. 3). Auch vermeintlich allgemeinverständliche Konzepte wie Arbeitsspeicher oder Dateipfade werden explizit erklärt. Zudem veranschaulichen wir potenziell unintuitive Techniken anhand alltagsentlehnter Metaphern – bei *Indexing* etwa über das Sinnbild nummerierter Häuser entlang einer Straße, wobei mithilfe des step-Parameters nur ungerade Hausnummern „angesprochen“ werden können (`musterstrasse[0:20:2]`). Unsere Lehrtätigkeit der vergangenen Jahre hat uns gezeigt, dass Materialien zur Vermittlung von Programmierkenntnissen an GSW-Studierende auf ganzer Linie niedrigschwellig ausgestaltet sein müssen, damit diese nicht abgeschreckt werden. Dass unsere Inhalte im Gegensatz zu den meisten anderen Programmiereinführungen deutschsprachig sind, trägt ebenso zu deren Zugänglichkeit für unsere Zielgruppe bei.

Abseits der konkreten Inhalte versuchen wir Lernende letztlich in dreifacher Hinsicht auf das Programmieren einzustimmen. Erstens motivieren wir direkt zu Beginn explizit den Nutzen von Programmierskills in den GSW, in dem wir anhand von Beispielen aufzeigen, was man mithilfe von Code in diesen Disziplinen erreichen kann, was mit einem händischen Ansatz nur unter hohem Zeitaufwand oder erst gar nicht umsetzbar wäre. Zweitens führen wir noch vor konkreten Python-Inhalten in Algorithmisches Denken ein. Dabei handelt es sich um die Fähigkeit, 1) ein gegebenes Problem präzise zu analysieren, 2) zielführende Schritte zu dessen Lösung zu finden, 3) diese Schritte zu operationalisieren, d. h. einen kompletten und korrekten Algorithmus zu formulieren, 4) diesen Algorithmus sowohl an typischen wie atyischen Fälle zu testen und 5) seine Effizienz zu verbessern [4, S. 95]. Die Verinnerlichung dieser grundsätzlichen Art der Herangehensweise an ein (Programmier-)Problem scheint nämlich wichtiger als die effektive Syntax und Semantik der jeweiligen Programmiersprache [4, S. 90]. Ergebnisse aus einem iterativ konzipierten Programmierkurs, in dem jeweils nur ein Parameter zum vorangegangenen Kurs angepasst wurde, legen nahe, dass das Voranstellen selbst einer kurzen Einheit zu algorithmischem Denken zu besseren Programmierfertigkeiten führt [9]. Dies deckt sich mit unserer Erfahrung, dass Lernende eine Art Aha-Effekt erleben, wenn sie sich diese vollends logische Art des Denkens bewusst machen. Drittens versuchen wir Enttäuschung und Abbrüchen proaktiv vorzubeugen, indem wir den Lernenden an diversen Stellen in unseren Notebooks deutlich machen, dass sie zwar häufig scheitern werden, dass dies aber wirklich ganz normal ist (s. Abb. 4).

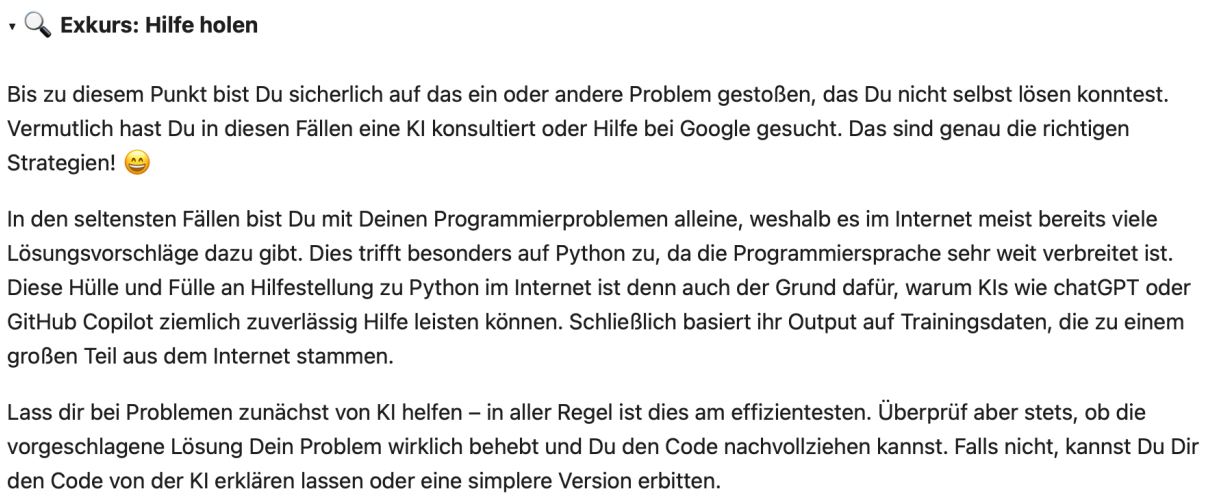


Abb. 2: Ausschnitt aus einem Notebook zum Umgang mit Fehlern und Problemen, in dem wir auch implizites Erwartungsmanagement betreiben.

Aufbauend auf den Grundlagennotebooks führen weitere, in sich abgeschlossene Notebooks in die Datenanalyse mit `pandas`, reguläre Ausdrücke, Web Scraping sowie Tagging (z. B. Lemmatisierung oder Sentiment Analysis) ein. So können sich Lernende aus den verschiedenen GSW modular einen für sie relevanten Programmierkurs zusammenstellen. Angenommen, eine Studentin möchte ein bereits existierendes Sprachkorpus mit zusätzlichen Informationen wie Wortarten oder syntaktischen Abhängigkeiten anreichern, so kann sie ihre Pythonkenntnisse gezielt im Bereich Tagging erweitern. Ihr Kommilitone, der seinen eigenen Datensatz erst aus dem Internet zusammenstellen will, kann sein Skill¬set dagegen mit Blick auf Web Scraping komplettieren.

Neben den eigentlichen Notebooks bieten wir umfangreiche Zusatzinhalte. Obschon die Lehrnotebooks bereits zahlreiche Übungen enthalten (s. Abb. 3), stellen wir zu jedem Thema weitere Übungen (und natürlich Lösungen) bereit, da wir die Erfahrung gemacht haben, dass Studierende durch nichts besser lernen als durch selbständige, kreative Anwendung der Theorie. Weiter bestehen Materialien u. a. zum Umgang mit der Command Line, mit Git(Hub), zur Visualisierung von Daten sowie zum Trainieren eines eigenen, basalen Sprachmodells.

Sämtliche Materialien sind wie erwähnt in einem eigenen GitHub-Repositorium abgelegt und als Open Educational Resources (*OER*) unter CC-BY-SA-Lizenz bei den gängigen Plattformen twillo und Wikiversity referenziert. Auch Lehrenden stehen sie zur Verfügung – dank der modularen Struktur können unsere Materialien sowohl *as-is* eingesetzt als auch flexibel um weitere Inhalte ergänzt werden.

4. Erprobung in Flipped Classroom-Seminaren

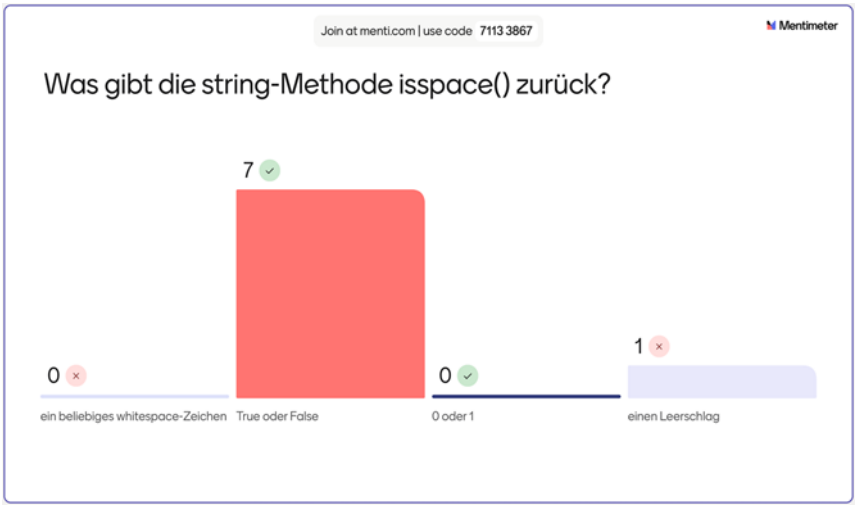
An der TU Dresden kommen die eben beschriebenen beschriebenen Materialien seit dem Wintersemester 2022/23 in einem Seminar zum Einsatz, das regulär von der Professur für Angewandte Linguistik angeboten wird. Im Sinne des Austausches zwischen den einzelnen *virTUos*-Teilprojekten haben wir zudem einmalig eine gemeinsame Lehrveranstaltung mit *DigitalHerrnhut* durchgeführt, in der wir Herrnhuter Texte unter Einsatz der parallel erlernten Programmierskills für die sprachhistorische Forschung aufbereiteten und analysierten.

Unser Seminar ist im Sinne von Blended Learning als Flipped-Classroom-Format konzipiert [10], [11], [12]. Asynchrone und synchrone Lernphasen sind didaktisch miteinander verzahnt, wobei Studierende die Notebooks primär asynchron – flexibel in Bezug auf Zeit, Ort und Tempo – bearbeiten und die synchronen, räumlich-kopräsenten Sitzungen der Rekapitulation sowie dem *gemeinsamen* Üben und Coden dienen (s. beispielhafter Seminarplan in

Datum	Zeit	Synchron	Asynchron (nach Sitzung)
16.10.	13:00-14:30	Installation & Algorithmisches Denken	„Einführung“
23.10.	13:00-14:30	„Einführung“ rekapitulieren	„Datentypen“
30.10.	13:00-14:30	„Datentypen“ rekapitulieren	„Kontrollstrukturen“
06.11.	13:00-14:30	„Kontrollstrukturen“ rekapitulieren	„Funktionen und Methoden Teil 1“
13.11.	13:00-14:30	„Funktionen und Methoden Teil 1“ rekapitulieren	„Funktionen und Methoden Teil 2“
27.11.	13:00-14:30	„Funktionen und Methoden Teil 2“ rekapitulieren	Vorbereitung Programmieren mit KI
04.12.	13:00-14:30	Programmieren mit KI	„Input und Output Teil 1“
11.12.	13:00-14:30	„Input und Output Teil 1“ rekapitulieren	„Input und Output Teil 2“
18.12	13:00-14:30	„Input und Output Teil 2“ rekapitulieren	„Datenanalyse Teil 1“
08.01.	13:00-14:30	„Datenanalyse Teil 1“ rekapitulieren	„Datenanalyse Teil 2“
15.01.	13:00-14:30	„Datenanalyse Teil 2“ rekapitulieren	*„Reguläre Ausdrücke“*
22.01.	13:00-14:30	Vorbereitung Mini-Hackathon	*„Tagging“*
29.01.	13:00-17:00	Mini-Hackathon	Vorbereitung Projektarbeit
05.02.	13:00-14:30	Vorbereitung Projektarbeit	

Abb. 3. Beispielhafter Seminarplan aus dem Wintersemester 2024/25, der die Verzahnung asynchroner und synchroner Lernphasen im Sinne des Flipped Classroom-Konzepts aufzeigt. Dieses Seminar wurde auch als Testbed für die Integration generativer KI genutzt (s. u.).

Abb. 5). Kollaborative Lernsettings haben sich in verschiedenen Studien als förderlich bei der Vermittlung von Programmierkenntnissen erwiesen [1], [9], [13]. In unserem Seminar löst der Dozent einerseits komplexere sowie besonders anschauliche Programmierprobleme frontal, jedoch stets unter Einbezug der Studierenden. Auch hier wird die inkrementelle und iterative Praxis des Codens deutlich, gleichzeitig wird darauf geachtet, ein fehlertolerantes Mindset zu kultivieren. Andererseits implementieren wir Settings, die Kollaboration unter den Studierenden fördern. So schaffen wir jeweils zu Beginn des Semesters *peer groups* bestehend aus zwei oder drei Studierenden – sofern möglich mit unterschiedlichen Vorkenntnissen –, die über das Seminar hinweg in den synchronen Sitzungen zusammenarbeiten sollen. Höhepunkt der Kollaboration unter Studierenden ist jeweils ein *Mini-Hackathon* am Ende des Semesters. Während vier Stunden widmen sich die Teilnehmenden einem vorher von ihnen selbst gewählten Datensatz, erarbeiten für sie interessante Fragestellungen und machen sich daran, diese mithilfe von Code zu operationalisieren, um am Ende des Mini-Hackathons ihre Ergebnisse zu präsentieren sowie den Prozess zu reflektieren. Solche Mini-Hackathons ermöglichen, dass Studierende Ideen und Lösungen kokonstruieren, die sie allein womöglich nicht entwickelt hätten [1]. Über die Semester hinweg wurden diverse Korpora wie stenografische Bundestagsprotokolle, User-Kommentare zum Instagram-Projekt *@ichbinsophiescholl* [14], Fussballspielstatistiken oder Zeitungstexte analysiert, ersterer Datensatz etwa im Hinblick auf die Verteilung von Zwischenrufen im Parlament nach Fraktionen sowie einzelnen Abgeordneten. Wenngleich die Studierenden ihre wachsenden Pythonkenntnisse während des gesamten Semesters in thematisch eher isolierten Übungen anwenden können, kommen diese beim Mini-Hackathon zum ersten Mal in größerem und ganzheitlicherem Umfang zum Einsatz. Vor allem aber wird die Codingpraxis dabei von den Interessen der Studierenden selbst geleitet, was sich positiv auf ihre Motivation auswirkt. Neben diesen kollaborativen Komponenten des Seminars erfreuen sich insbesondere gamifizierte Aspekte wie Wissensquizes großer Beliebtheit, da diese eine Auflockerung während der Sitzung bieten sowie scheinbar wegen ihres kompetitiven Charakters (s. Abb. 6). Ihre finale Anwendung finden die neu erlernten Programmierskills schließlich in den an das Seminar anschließenden Prüfungsleistungen (sog. *Projektarbeiten*). Typischerweise absolvieren Studierende dafür ihr eigenes kleines Forschungsprojekt – von der Datenerschließung (z. B. mithilfe von Web Scraping), über die Datenaufbereitung (z. B. mithilfe regulärer Ausdrücken) und -auswertung (z. B. mit `pandas`) hin zur Visualisierung der Ergebnisse (z. B. mit `matplotlib`). Viele



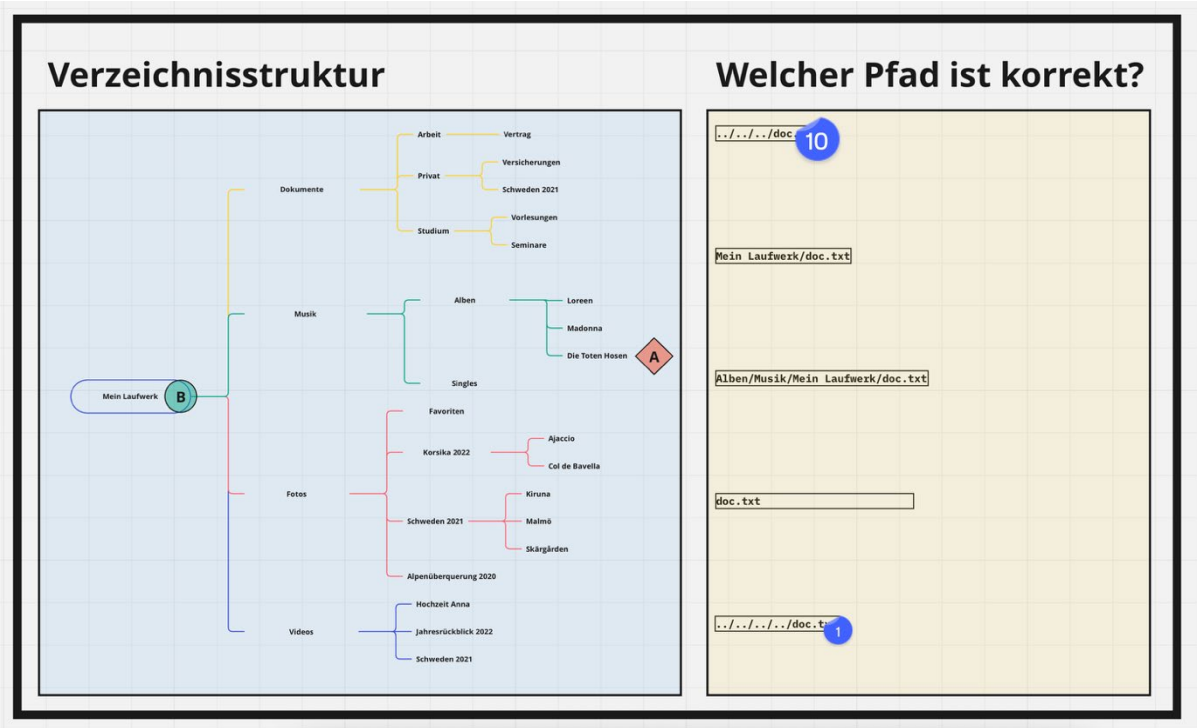


Abb. 4: Gamifizierte Seminarkomponenten: Wissensabfrage zu string-Methoden (oben; umgesetzt mit mentimeter.com) bzw. Überprüfung des Verständnisses relativer Dateipfade, um von einem gegebenen Verzeichnis A in ein Verzeichnis B zu navigieren (unten; miro.com).

Studierende berichten, dass sie durch ihre Projekte noch einmal einen großen Kompetenzsprung gemacht haben, auch weil sie sich nun von ihrem ganz individuellen Erkenntnisinteresse angetrieben komplexerem Code zuwandten. Über die Semester hinweg sind viele hochqualitative Prüfungsleistungen eingereicht worden, u. a. eine linguistische zu Helvetismen und Austriazismen in deutschsprachigen Zeitungstexten, eine geschichtswissenschaftliche zur Kartierung des digitalen Herrnhuter Gottesacker sowie eine bildwissenschaftliche, die historische Massenproduktionsstrategien mit digitalen Reproduktionsverfahren verglich.

Das Flipped Classroom-Format unseres Seminars, das fundamental an unser übergreifendes Selbstlernkonzept anschließt (s. o.), hat sich in den letzten sechs Semester bewährt. Manche Studierende waren zwar mitunter überfordert, den Großteil der Lerninhalte in Eigenregie zu bearbeiten, die Mehrheit der Teilnehmenden schätzte das Format jedoch oder erkannte zumindest pragmatisch an, dass man Programmieren nur wirklich lernen kann, indem man codet, codet und nochmal codet –

und das muss man schlicht *selbständig* tun. Abgesehen davon, dass wöchentliche Sitzungen hierfür nicht ausreichend wären, kann einem dies weder von der Lehrperson noch vom kollaborativem Austausch mit *peers* abgenommen werden. Diese in den synchronen Anteilen des Seminars verankerten Elemente bieten eine wichtige Unterstützung für den individuellen Lernprozess, können diesen aber nicht ersetzen. Gleichzeitig hat sich in den vergangenen Jahren gezeigt, dass die synchrone Sitzung bei komplexeren Themen gegen Ende des Seminars durchaus mehr als eine Doppelstunde umfassen sollte, allein weil Übungen zu diesem Zeitpunkt wesentlich zeitaufwendiger sind als jene zu den einführenden Inhalten. Ein komplettes Blockseminar, das sämtliche synchronen Anteile in wenigen, ganztägigen Einheiten bündelt, hat sich dagegen nicht bewährt, da die mentalen Aufnahmekapazitäten der Studierenden begrenzt sind und sich die so zentrale selbständige Programmierpraxis weniger zu etablieren scheint als in einem Kurs, der sich über das ganze Semester streckt.

Das beschriebene Seminar ist mittlerweile curricular im neuen Masterstudiengang *Digital Humanities* an der TU Dresden verankert, aber auch Studierende anderer Fachrichtungen einschließlich diverser Lehramtsstudiengänge nehmen das Angebot regelmäßig wahr. Das Seminar wird laufend verbessert, nicht zuletzt im Hinblick auf generative KI. Deren Aufkommen inmitten des Projektverlaufs warf neue Fragen auf und machte Anpassungen erforderlich, die im Folgenden beschrieben werden.

5. Generative KI

Gegenwärtige generative KI-Tools wie *chatGPT* oder *Google Gemini* sind mittlerweile in der Lage, Programmiercode vollautomatisch und zumeist fehlerfrei zu generieren [15]. Nachdem sich dies abzeichnete, stellte sich uns die grundlegende Frage, ob sich Programmierenlernen für GSW-Studierende überhaupt noch lohnt. Aus drei Gründen waren wir grundsätzlich davon überzeugt, dass Programmierkenntnisse in den besagten Disziplinen angesichts leistungsstarker KI nicht überflüssig, sondern im Gegenteil vielleicht relevanter denn je werden. Erstens zeigte die bisherige Erfahrung, dass unsere Zielgruppe aufgrund der eingangs beschriebenen technischen Hemmungen sowie generell geringer Digitalkompetenzen selten Code – auch KI-generierten – als Problemlösungsstrategie in Betracht zieht. Vor dem Hintergrund des digital turn in den GSW dürfte daher weiterhin ein niedrigschwelliger Einstieg notwendig sein, damit Studierende mit den dadurch relevant gewordenen Analysemethoden vertraut gemacht werden [8]. Zweitens kann generative KI nur dann genutzt werden, wenn sinnvolle Prompts formuliert werden. Auf Programmierung bezogenes *prompt engineering* setzt jedoch ein Verständnis dessen voraus, wann Code überhaupt sinnvoll ist, was Code (nicht) leisten kann, sowie Kenntnisse der einschlägigen Terminologie. Dieselben Skills benötigen Lernende, um generierten Output einschätzen und bei fehlerhaftem KI-Code agieren zu können. Auch diese Kompetenzen bedürfen nach wie vor zielgruppenadäquater Vermittlung. Drittens muss generierter Code immer auch ausgeführt werden. Zwar ermöglichen manche KI-Tools die Ausführung von Code in virtuellen Umgebungen. Sobald aber mit lokalen Dateien gearbeitet wird, was beim Programmieren zu Forschungszwecken bald unumgänglich wird, bedarf es einer eigenen *funktionierenden* Entwicklungsumgebung (*IDE*), für die der generierte Code zumeist adaptiert werden muss, z. B. im Hinblick auf Dateipfade. Das Einrichten einer solchen IDE stellt unserer Erfahrung nach aber eine große Einstiegshürde dar, die es weiterhin über entsprechende Lernangebote abzubauen gilt.

Wenn Programmierkenntnisse in den GSW und deren Vermittlung in der Hochschullehre durch generative KI also nicht obsolet geworden sind, drängt sich die Anschlussfrage auf, wie ein Lehr-/Lernformat wie unseres an die neuen Rahmenbedingungen angepasst werden muss, die KI für das Programmieren stiftet. Um eine sinnvolle Integration von KI zu explorieren, nutzten wir unser Seminar im Wintersemester 2024/25 als *Testbed*.

Eine Umfrage zu Seminarbeginn ergab, dass Studierende generative KI zwar vielseitig nutzten, aber nicht zum Programmieren. Als Gründe wurden wie von uns antizipiert fehlende Basics zum Prompten und zur Einschätzung des Outputs sowie die Angst vor Fehlern

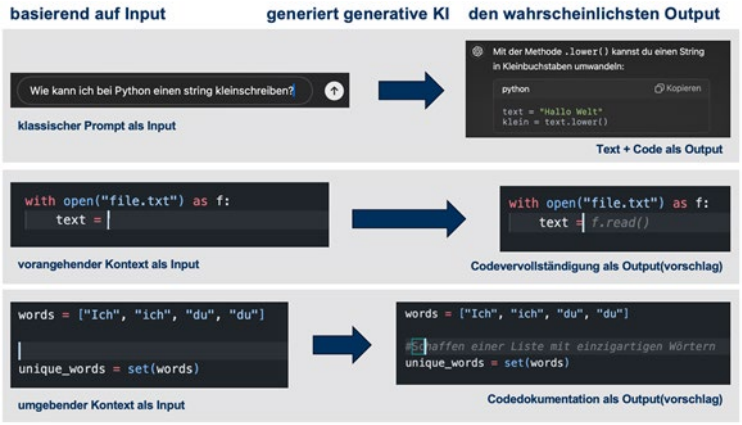


Abb. 5: Theoretischer Input zur Funktionsweise generativer KI.

im KI-Code genannt. Da wir annahmen, dass gewisse Grundkenntnisse vorhanden sein müssen, bevor KI gewinnbringend zum Programmieren eingesetzt werden kann, terminierten wir eine entsprechende Einführung erst in der Mitte des Semesters (s. Seminarplan des Testbedseminars in Abb. 5). Dabei stellten wir zunächst über einen theoretischen Input sicher, dass alle Teilnehmenden auf demselben Verständnisniveau bezüglich der Funktionsweise von generativer KI sind: Basierend auf einem Input – egal ob in Form von einem klassischen Prompt oder Codekontext – schlägt die KI den wahrscheinlichsten Output vor (s. Abb. 7). Weiter demonstrierten wir die probabilistische Basis von generativer menschlicher bzw. künstlicher Intelligenz anhand von natürlichsprachlichen Lückentexten bzw. unvollständigem Code, die die Studierenden individuell vervollständigen sollten. Wenig überraschend zeigte sich je nach „Input", dass gewisse „Outputs" wahrscheinlicher als andere sind, genau wie das auch bei generativer KI „hinter den Kulissen" der Fall ist (s. Abb. 8).

Anschließend differenzierten wir zwischen vier Anwendungsfällen, bei denen generative KI konstruktiv eingesetzt werden kann: Codegeneration „from scratch", Fehlerbehebung, Kommentierung bzw. Dokumentation von Code und Sich-erklären-lassen von Code. Jeder Typ wurde dann anhand von Beispielen und unter Einsatz sowohl „allgemeiner" KI wie *chatGPT* als auch fürs Coden optimierter KI, *GitHub Copilot*, geübt. Im weiteren Verlauf des Seminars sowie beim Mini-Hackathon und den Prüfungsleistungen war es den Studierenden freigestellt, ob und wie sie generative KI einsetzten. Der Dozent demonstrierte fortlaufend ihren Einsatz in den synchronen Sitzungen bei der gemeinsamen Bearbeitung von Übungen.

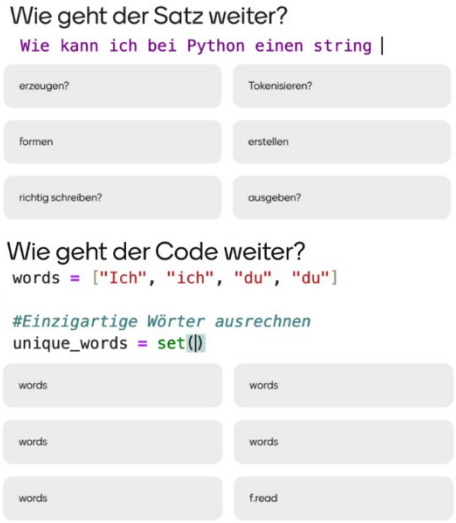


Abb. 6: Sensibilisierung für die probabilistische Basis generativer KI. Studierende sollten den Satz/Code individuell vervollständigen. Der Screenshot zeigt einige ihrer Antworten.

Nach Abschluss des Seminars wurde die getestete KI-Integration über extern moderierte Fokusgruppen evaluiert, um praktische Erfahrungen der Studierenden einzuholen [5]. Demnach bewerteten Studierende sowohl die Art als auch den Zeitpunkt der Einführung Mitte Semester als sehr sinnvoll und schätzten ebenso die Selbstbestimmung bei der Toolnutzung. Die meisten Studierenden berichteten, dass sie ohne unsere niedrigschwellige Einführung nie mit Programmieren begonnen hätten, sich nun aber imstande fühlten, KI kompetent dazu einzusetzen. Als primäre Einsatzfelder von KI stellten sich die Korrektur von syntaktischen Fehlern sowie die Codedokumentation

heraus. Ebenfalls lassen sich Studierende Code gerne „herunterbrechen“, also erklären. Gleichzeitig wurde berichtet, dass vollständig KI-generierte Codes häufig zu komplex sind und z. B. über Abhängigkeiten neue Probleme schaffen. Nicht zuletzt deshalb dauere die Interaktion mit der KI manchmal länger als das Finden einer eigenen Lösung. Divergierende Einstellungen mit Blick auf KI-Nutzung in peer groups sowie individuelle Hemmungen (Selbstansprüche; Angst vor Fehlern; Angst vor eigenem Kreativitätsverlust; das Gefühl, zu betrügen) scheinen einem (weiteren) Einsatz generativer KI beim Programmieren ebenso im Weg zu stehen. Interessant war zudem die mehrfach getätigte Aussage, dass Studierende lieber den Dozenten als die KI fragten, da dieser menschlicher reagiere, indem er seine Antworten in den realen Seminarkontext einbettet.

Es ist unstrittig, dass Code eine zentrale wissenschaftliche Problemlösungsstrategie bleibt, auch in den zunehmend digitalen GSW. Generative KI hat sich in unserem Testseminar als hilfreicher Katalysator beim Erlernen und Anwenden von Programmierkenntnissen erwiesen. Wie von uns angenommen, zeigte sich jedoch deutlich, dass zunächst Grundkenntnisse vorhanden sein müssen, damit Lernende in ihrer Programmierpraxis überhaupt von generativer KI profitieren können. Es bestätigte sich folglich, dass eine niedrigschwellige, zielgruppensensible Vermittlung von Programmierskills in der Hochschullehre gerade in Zeiten von leistungsstarker KI hochrelevant ist. Nur so können die eingangs beschriebenen Potenziale des digital turn effektiv ausgenutzt werden. Entsprechend wurde die getestete Form der KI-Integration ins Seminar auch im Sommersemester 2025 beibehalten. Auch wurden die Lehr-/Lernmaterialien dahingehend überarbeitet, dass an sinnvollen Stellen auf die Möglichkeit von KI-Nutzung hingewiesen wird (s. auch Abb. 4).

6. Ausblick

Anspruch von *ExDiMed* war es, Lernende aus den verschiedenen GSW mit keinerlei Vorkenntnissen durch unsere Inhalte sowie deren Anwendung in Seminaren in die Lage zu versetzen, geistes- und sozialwissenschaftliche Fragestellungen auf digital kompetente Weise zu bearbeiten – vom Beschaffen eigener Daten über die Datenaufbereitung bis hin zur Auswertung und Visualisierung. Zahlreiche gute Prüfungsleistungen ebenso wie weiterführende Arbeiten (z. B. programmierintensive Masterarbeiten) und das Feedback der Lernenden während sechs Semestern zeigen uns, dass wir diesem Ziel zumindest für die GSW an der TU Dresden ein gutes Stück nähergekommen sind.

Während sowohl die Erstellung der Inhalte als auch die Seminarleitung bislang stark durch einen einzigen wissenschaftlichen Mitarbeiter geleistet wurde, findet gegenwärtig eine Übergabe an neue Lehrpersonen statt, die sowohl die Materialien als auch die Lehrveranstaltung nach Projektende weiterführen werden. Dazu werden zurzeit sämtliche Prozesse, etwa bei der Handhabung des GitHub-Repositoriums, dokumentiert. Neben der Erstellung neuer Aufbaumodule steht künftig insbesondere das laufende Onboarding neuer Entwicklungen bei generativer KI im Fokus. Über die TU Dresden hinaus wünschen wir uns, dass unsere Materialien von Dozierenden aus den GSW an anderen deutschsprachigen Hochschulen in ihrer Lehre eingesetzt werden, sei es in ihrer Gesamtheit, zur Vermittlung der Grundlagen für anschließende fachspezifische Inhalte oder unter Einsatz einzelner Aufbaumodule ergänzend zu einem bestehenden Kurs.

Danksagung

Das Projekt *ExDiMed* wird von der *Stiftung Innovation in der Hochschullehre* im Rahmen des *virTUos*-Verbunds gefördert.

Literatur

[1] L. Beck und A. Chizhik, „Cooperative learning instructional methods for CS1: Design, implementation, and evaluation“, ACM Trans. Comput. Educ., Bd. 13, Nr. 3, S. 10:1-10:21, Aug. 2013, doi: 10.1145/2492686.

[2] H. Snee, C. Hine, Y. Morey, S. Roberts, und H. Watson, „Digital Methods as Mainstream Methodology: An Introduction“, in Digital Methods for Social Science: An Interdisciplinary Guide to Research Innovation, H. Snee, C. Hine, Y. Morey, S. Roberts, und H. Watson, Hrsg., London: Palgrave Macmillan UK, 2016, S. 1–11. doi: 10.1057/9781137453662_1.

[3] Deutsche Forschungsgemeinschaft, „Digitaler Wandel in den Wissenschaften“, Okt. 2020, doi: 10.5281/ZENODO.4191345.

[4] F. Jannidis, H. Kohle, und M. Rehbein, Hrsg., Digital Humanities. Eine Einführung. Stuttgart: Metzler, 2017.

[5] Y. Frommherz, A. J. Matz, und S. Meier-Vieracker, „KI-gestütztes Programmierenlernen – Erprobung und Erfahrungen eines Lehr-Lernkonzepts“. Zugegriffen: 14. August 2025. [Online]. Verfügbar unter: https://fis.uni-bamberg.de/handle/uniba/107576

[6] G. Punziano, „Adaptive Epistemology: Embracing Generative AI as a Paradigm Shift in Social Science“, Societies, Bd. 15, Nr. 7, S. 205, Juli 2025, doi: 10.3390/soc15070205.

[7] S. Schiller-Stoff, L. E. Münzer, C. Dittmann, und S. Sagadin, „Der Einfluss von AI-Pair-Programmers auf die Digital Humanities: Potentiale und Limitationen“, in Book of Abstracts DHd 2025, Bielefeld, 2025. doi: https://doi.org/10.5281/zenodo.14887461.

[8] Y. Frommherz und J. Langenhorst, „Digitale Kompetenzen für Geistes- und Sozialwissenschaftler:innen. Vorzüge eines Blended Learning-Formats für die Vermittlung von Programmierkenntnissen.“, Lessons Learn., Bd. 2, Nr. 1, Juli 2022, doi: 10.25369/ll.v2i1.37.

[9] T. Koulouri, S. Lauria, und R. D. Macredie, „Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches“, ACM Trans. Comput. Educ., Bd. 14, Nr. 4, S. 1–28, Feb. 2015, doi: 10.1145/2662412.

[10] S. Seufert und P. Mayr, Fachlexikon e-lerning: Wegweiser durch das e-Vokabular. in managerSeminare. Bonn: ManagerSeminare May, 2002.

[11] E. Schoop, H. Bukvova, und C. Lieske, „Blended-Learning arrangements for higher education in the changing knowledge society“, in Proceedings of the International Conference on Current Issues in Management of Business and Society Development, Riga, Lat., Dresden: TU Dresden, 2010. [Online]. Verfügbar unter: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-26183

[12] M. Kerres, Mediendidaktik: Konzeption und Entwicklung mediengestützter Lernangebote. München: OLDENBOURG WISSENSCHAFTSVERLAG, 2013. doi: 10.1524/9783486736038.

[13] G. Braught, T. Wahls, und L. M. Eby, „The Case for Pair Programming in the Computer Science Classroom“, ACM Trans. Comput. Educ., Bd. 11, Nr. 1, S. 2:1-2:21, Feb. 2011, doi: 10.1145/1921607.1921609.

[14] S. Meier-Vieracker, „‚LIEBE SOPHIE‘ – ADRESSIERUNG UND INVOLVIERUNG IN INSTAGRAM-KOMMENTAREN AM BEISPIEL DES PROJEKTES @ICHBINSOPHIESCHOLL“, Dez. 2023, doi: 10.48694/KORDAF.3849.

[15] A. Ziegler u. a., „Productivity assessment of neural code completion“, in Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego CA USA: ACM, Juni 2022, S. 21–29. doi: 10.1145/3520312.3534864.