Verändert vertieftes Verständnis von LLMs die Codequalität?

Verändert vertieftes Verständnis von LLMs die Codequalität? Studienlage zu einer Nullhypothese

Nullhypothese: Zwischen Programmierern mit und ohne vertieftes Verständnis von großen Sprachmodellen (LLMs), die mit diesen programmieren, gibt es keine Unterschiede in der Ergebnisqualität des Codes.

Review erstellt am 30.9.2026 von Qwen Code (KI-Agent, LLM: glm-5.3-flash). Literatur-Identifikation ausschließlich über OpenAlex; 12 von 14 Kernstudien wurden als Volltext ausgewertet, 2 nur als Abstract (kenntlich gemacht). Seitenangaben beruhen auf der PDF-Textlage; bei aus dem Zotero-Index gelesenen Texten sind sie näherungsweise („S. ~“).

1. Fragestellung und Befundlage

Die Nullhypothese behauptet einen Null-Effekt: Die Ergebnisqualität des Codes — Korrektheit, Sicherheit, Wartbarkeit, Aufgabenerfolg — soll unabhängig davon sein, wie tief ein Programmierer das Werkzeug versteht, mit dem er arbeitet. Eine Studie, die „vertieftes LLM-Verständnis” direkt manipuliert (Schulung ja/nein) und anschließend die Codequalität misst, existiert in der aufgefundenen Literatur nicht. Die Evidenzlage zerfällt daher in vier indirekte Linien:

1. **Zufallsexperimente KI-Assistenz gegen Kontrollgruppe** (misst den KI-Effekt, nicht den Verständnis-Effekt, erlaubt aber Blick auf Streuung innerhalb der KI-Gruppe),
2. **Vergleiche zwischen Kompetenzgruppen unter KI-Bedingung** (stark gegen schwach, Experte gegen Novize),
3. **Beobachtungs- und Interviewstudien** zu Interaktionsmustern, Vertrauen und Verifikationsverhalten,
4. **Benchmark- und Kalibrierungsbefunde** (was die Modelle leisten und wie gut Nutzer das einschätzen).

„Vertieftes Verständnis von LLMs” wird in diesen Studien nie direkt gemessen. Als Näherungsgrößen (Proxies) dienen: Prompt-Kompetenz und Prompt-Verhalten, Vorerfahrung mit KI-Codegeneratoren, fachliche Programmierexpertise,

beobachtete Verifikations- und Testpraktiken sowie die Passung von Mental Models (inneren Arbeitsmodellen) an die tatsächliche Funktionsweise der Modelle.

2. Suchstrategie

Datenbank: OpenAlex (api.openalex.org), Abfragedatum 30.9.2026, Filter `from_publication_date:2019-01-01`, `is_retracted:false`. Kein Treffer wurde aus dem Web oder Vorwissen identifiziert; Vorwissen diente nur der Formulierung der Konzeptblöcke.

Konzeptblöcke: (a) LLM-Verständnis/Kompetenz: „LLM literacy", „AI literacy", prompt engineering + expertise, mental model; (b) Programmieren mit LLMs: AI-assisted programming, code generation, GitHub Copilot, vibe coding; (c) Outcome: code quality, correctness, security vulnerabilities, maintainability, skill formation.

Queries (Auswahl, je 15–25 Top-Treffer gesichtet): „AI literacy"+„code quality"; „LLM literacy"; „prompt engineering"+expertise+programming; „GitHub Copilot"+(„code quality" OR correctness OR vulnerabilit); „AI-assisted programming"+expertise+„code quality"; „mental model"+„code generation"+(programmer OR novice OR expert); overreliance; „insecure code"+„AI assistant"; „prompt engineering"+„code quality"+experiment; „widening gap"+novice programmers+„generative AI"; „vibe coding"+(experience OR expertise); „developer expertise"+„code generation"+LLM; dazu Autorenfilter (Mozannar) und Titelprüfungen (Satisficing, METR).

Schneeball: Vorwärtsschneeball von Perry et al. 2023 (178 zitierende Werke seit 6/2024 gesichtet, sortiert nach Zitathäufigkeit); Rückwärtsschneeball aus den Literaturverzeichnissen der Volltexte (u. a. Peng et al., Kazemitabaar-Kaskade, Xue et al., Jost et al.).

Auswahl: Aus rund 15.000 Roh Treffern der Queries wurden 14 Kernstudien in die Zotero-Sammlung „Programmieren mit KI" (9NP25QCP) importiert: Kriterien waren (i) direkter Bezug zur Moderationsfrage (Nutzerkompetenz als Faktor), (ii) experimentelles oder systematisch-empirisches Design, (iii) messbarer Code- oder Kompetenzbezug. Zitat-/Reputationszahlen (234 Zitate für Perry, 890 für „Johnny Can't Prompt", 404 für „Grounded Copilot", Stand 30.9.2026) dienten nur der Ordnung, nicht der Aufnahme.

3. Studienübersicht

#	Studie	Design, N, Population	Outcome und Einordnung	Volltext
1	Perry et al. 2023, CCS	Randomisiert (KI vs. ohne), N=54; Studierende bis Profis	Sicherheitslücken, Korrektheit — Experiment	ja
2	Kazemitabaar et al. 2023, ICER	Matched groups (Codex vs. ohne), N=69; Python-Novizen	Korrektheit, Zeit, Retention — Experiment	ja

#	Studie	Design, N, Population	Outcome und Einordnung	Volltext
3	Prather et al. 2024, ICER („Widening Gap“)	Lab-Experiment + Interviews, N=21; Studierende (C++)	Aufgabenlösung, Verhalten — Experiment, kleine N	ja
4	Zamfirescu-Pereira et al. 2023, CHI („Johnny Can’t Prompt“)	User Study, N=19; Domänenexperten ohne KI-Erfahrung	Prompt-Erfolg — Beobachtung	nein (nur Abstract)
5	Wermelinger 2023, SIGCSE	Fallserie, 8 Aufgaben; Copilot gegen Referenzlösungen	Lösungsrate, Korrektheit — Fallserie	nein (nur Abstract)
6	Barke et al. 2023, TOPLAS („Grounded Copilot“)	Beobachtungsstudie, N=20; Profis bis Studierende	Interaktionsmodi, Validierung	ja
7	Liu, Sarkar & Gordon 2023, CHI	Explorativ + Intervention, ca. 20; End-User-Programmierer	Mental Models, Vertrauen	ja
8	Nguyen et al. 2024, CHI („(Mis)read Each Other“)	Experiment, N=120; nahe-Novizen	Prompt-Erfolg, Lösungsrate — Experiment	ja
9	Moradi Dakhel et al. 2023, JSS	Benchmark Copilot gegen Mensch, 100+ Lösungen	Korrektheit, Reparaturaufwand	ja (arXiv-Fassung)
10	Becker et al. 2025, METR	RCT in der Praxis, N=16, 246 Tasks; erfahrene OSS-Entwickler	Fertigstellungszeit, Perzeption — RCT	ja
11	Fawzy et al. 2026	Survey, N=162; drei Erfahrungsgruppen	Praktiken, Perzeptionen — Querschnitt	ja
12	Shen & Tamkin 2026	Randomisierte Experimente; neue Python-Library	Verständnis, Debugging — Experiment	ja
13	Wang et al. 2024, FAccT	Interviews N=17 + Design-Probe N=12; Profi-Entwickler	Vertrauensurteile — Beobachtung	ja
14	Borg et al. 2024	Registered Report Stage 1; Profis (geplant)	Wartbarkeit — Protokoll ohne Ergebnisse	ja

4. Befunde entlang der vier Evidenzlinien

4.1 Zufallsexperimente: Was passiert, wenn KI ins Spiel kommt?

Perry et al. 2023 ist die am direktesten einschlägige Arbeit. In einem randomisierten Experiment lösten 54 Teilnehmer (Studierende bis Berufstätige mit

Jahrzehnten Erfahrung) kryptografische Programmieraufgaben — Experimentalgruppe mit KI-Assistent, Kontrollgruppe ohne (S. 3). Die KI-Gruppe schrieb deutlich sicherheitskritischeren Code; bemerkenswert: die Teilnehmer mit KI hielten ihren Code für sicherer und vertrauten dem Assistenten mehr als die Kontrollgruppe, obwohl ihr Code objektiv unsicherer war (S. 8). Für die eigentliche Moderationsfrage ist der Diskussionsteil entscheidend: Teilnehmer, die **proaktiv Parameter anpassten und Prompts umformulierten**, lieferten häufiger korrekten und sicheren Code — die Autoren sprechen von sich ausbildenden „mental models" des Assistenten (S. 11). Wer die KI also besser zu lesen verstand, produzierte messbar besseren Code. Grenzen: N=54, eine Aufgabenfamilie (Sicherheit), die Prompt-Verhaltensanalyse ist explizativ, nicht konfirmatorisch.

Kazemitabaar et al. 2023 setzten 69 jugendliche Python-Novizen in einem matched-groups-Design mit bzw. ohne Codex ein. Die Korrektheit der Aufgabenlösungen unterschied sich insgesamt wenig; bei Arrays punktete die Codex-Gruppe (+20,8 Prozent, p=.022, S. 10). Der Retentionstest eine Woche später **ohne** KI zeigte keine signifikanten Nachteile der Codex-Gruppe — die befürchtete Deformation der Fähigkeiten trat in diesem kurzen Zeitraum nicht ein (S. 10). Aufschlussreich für die Moderationsfrage: Die Effekte der KI konzentrierten sich auf die leistungsstärkere Hälfte — die Codex-High-Gruppe schlug die Baseline-High-Gruppe auf mehreren Maßen deutlich, während zwischen den schwachen Gruppen kaum Unterschiede bestanden (S. 11). Bereits Kompetentere zogen also mehr Nutzen. Begrenzung: kurze Interventionsdauer, Kind/Jugend-Population.

Shen & Tamkin 2026 lernten professionelle Programmierer eine neue asynchrone Python-Bibliothek — mit oder ohne KI-Assistenz. Das Ergebnis kehrt die übliche Produktivitätserzählung um: KI-Nutzung **beeinträchtigte** konzeptuelles Verständnis, Code-Lesen und Debugging-Fähigkeit, ohne im Durchschnitt signifikante Effizienzgewinne zu bringen (S. 1). Wer vollständig delegierte, arbeitete zwar etwas schneller, lernte aber die Bibliothek nicht (S. 1). Der für die Nullhypothese wichtigste Befund: Die Autoren identifizierten **sechs unterscheidbare KI-Interaktionsmuster, von denen drei kognitive Einbindung zeigen und Lernergebnisse auch unter KI-Nutzung erhalten** (S. 1). Die Art, wie man die KI benutzt — tief einbindend oder delegierend — entscheidet also darüber, was beim Nutzer ankommt. Grenze: eine Stunde Lernzeit, gemessen wurde Kompetenzerwerb, nicht unmittelbare Codequalität; Prompt-Flüssigkeit wurde nicht erhoben (S. 23).

Becker et al. 2025 (METR) messen keine Codequalität, aber etwas, das die Frage rahmt: In einem RCT mit 16 erfahrenen Open-Source-Entwicklern an 246 echten Tasks in vertrauten, zehn Jahre alten Repositorien verlangsamte KI-Nutzung die Fertigstellung um **19 Prozent** — während die Teilnehmer vorab −24 Prozent erwarteten und auch nach der Teilnahme noch −20 Prozent schätzten (S. 2, S. 10). Auch externe Experten (Maschinelles-Lernen- und Ökonomie-Forscher) überschätzten den Nutzen massiv (S. 2). Wer sein KI-Verständnis an Selbstauskunft

messen wollte, läge hier um rund 40 Prozentpunkte daneben. Das ist das stärkste Einzelargument dafür, dass „Verständnis von LLMs" eine verlässliche Größe ist, die man nicht einfach voraussetzen darf — Selbstsicherheit und Verständnis fallen auseinander. Grenze: Cursor Pro mit Claude 3.5/3.7 (Stand Frühjahr 2025), Zeit als alleiniges Outcome, N=16.

4.2 Kompetenzgruppen unter KI-Bedingung: Spreizung statt Gleichheit

Prather et al. 2024 replizierten ein klassisches Programmierlern-Experiment unter GenAI-Bedingungen (21 Studierende, C++, S. 4). Das Bild: Studierende mit besseren Noten und höherer Selbstwirksamkeit nutzten die Werkzeuge, um sich zügig zur Lösung zu bewegen; schwächere kämpften — und hielten dabei eine **„unwarranted illusion of competence"** aufrecht, ein unberechtigtes Kompetenzgefühl (S. 2). Zeitgewinn durch GenAI war messbar keiner vorhanden, obwohl viele Teilnehmer ihn wahrten (S. 21). Das Fazit der Autoren: Ohne explizite Vermittlung metakognitiver Fähigkeiten weitet sich die Kluft zwischen gut und schlecht vorbereiteten Lernenden (S. 21). Zur Nullhypothese: Die Ergebnisqualität unter KI-Bedingung spaltete sich entlang vorhandener Kompetenz — ein Befund gegen Gleichheit. Grenzen: N=21, eine Site (S. 6).

Moradi Dakhel et al. 2023 verglichen Copilot-Generierungen mit menschlichen Lösungen fundamentaler Algorithmenaufgaben. Die korrekte Rate der Menschen lag höher; fehlerhafte Copilot-Vorschläge waren allerdings mit weniger Aufwand reparierbar (S. 1). Zwei Befunde tragen unmittelbar zur Frage bei: Erstens hing die Qualität der Generierung **„greatly" von Knappheit und Tiefe des vom Entwickler gelieferten Prompts ab** (S. 25) — der Prompt ist hier eine Kompetenzvariable. Zweitens die explizite Schlussfolgerung: Copilot wird zum Asset, wenn erfahrene Entwickler ihn nutzen (deren Filterfähigkeit die Vorschläge prüft), und zur Liability bei Novizen, die fehlerhafte oder suboptimale Lösungen mangels Expertise nicht herausfiltern (S. 1). Grenzen: Algorithmenaufgaben, keine Nutzer-RCT — die Expertenaussage ist Interpretation der Autoren, kein innerhalb der Studie gemessener Effekt.

Nguyen et al. 2024 isolierten in einem großen Experiment (120 Studierende, 48 Aufgaben) das reine Prompt-Schreiben für Code-LLMs. Effektives Prompten erforderte Fähigkeiten, die Anfänger nicht besitzen; das Prompt-Schreiben blieb durchgängig die Barriere (S. 15). Auch die Rückübersicht versagte beidseitig: Beginnende Programmierer und Code-LLMs lasen sich gegenseitig falsch (Titelprogramm, S. 2). Die Autoren schließen, dass „technical communication and code understanding" weiter gelehrt werden müssen (S. 15).

Fawzy et al. 2026 befragten 162 „Vibe Coder" in drei Erfahrungsgruppen (Non-Coder, Novizen, Profis). Die Wahrnehmungen der Codequalität ähnelten sich über alle Gruppen; divergent waren Interaktionsstile und Qualitätssicherungspraktiken:

Profis dialogisieren iterativ und liefern reicheren Kontext, Non-Coder lassen die KI führen (S. 1, S. 24). Zwei Befunde sind für die Nullhypothese genauenswert: Erstens unterscheiden sich **strukturierte Planung und verifikationsorientiertes Prompten nicht signifikant nach Erfahrungsgruppe** ($p \geq .177$, vernachlässigbare Effektstärken, S. 17) — in dieser Dimension ein Pro-Nullhypothese-Befund. Zweitens beschreiben die Autoren eine **Perception–Action-Lücke**: Selbst wer Grenzen des KI-Codes erkennt, fehlen oft die Fähigkeiten, danach zu handeln; die Bewertung von KI-generiertem Code sei eine eigenständige Fertigkeit (S. 25).

4.3 Interaktion, Vertrauen, Mental Models

Barke et al. 2023 beobachteten 20 Programmierer bei der Arbeit mit Copilot und entwickelten daraus eine Grounded Theory mit zwei Interaktionsmodi: **Acceleration** (Vorschlag annehmen, Takt halten) und **Exploration** (die KI als Werkzeug zum Ausloten unbekannter APIs und Ansätze). Das Verhältnis der Modi hing systematisch von Sprachexpertise und Copilot-Vorerfahrung ab (S. 15); Novizen erwarteten überzogen, dass die KI high-level explorative Aufgaben löse (S. 16), und die Validierungsstrategien — Zeilenexamination, Ausführen, Dokumentation — verteilten sich je nach Modus und Person unterschiedlich (S. 17). Ein Protokollausschnitt zeigt das Qualitätsrisiko: Teilnehmer P9 scheiterte an einem subtil fehlerhaften Vorschlag, weil er den Fehler nicht lokalisieren konnte (S. 15). Die Studie liefert den Mechanismus, keine Effektstärke: Wie jemand die KI versteht, steuert, wo sie in seinen Arbeitsfluss eintritt — und damit, welche Fehler entstehen.

Liu, Sarkar & Gordon 2023 arbeiteten das „Abstraktions-Matching"-Problem heraus: End-User-Programmierer wissen nicht, auf welcher Abstraktionsebene das Modell „denn nun reagiert" („What it wants me to say"). Stichprobenartiges Sanity-Checking hielten Teilnehmer selbst für unzureichend bei großen Datenmengen (S. 27). Ihre Designintervention (systematisch generierte, „grounded" Beispiele) führte zu konsistenteren Mental Models, mehr Zuversicht und besserem Vertrauen (S. 18) — Verständnis ist demnach trainierbar, und sein Zuwachs verändert das Nutzungsverhalten.

Wang et al. 2024 interviewten 17 professionelle Nutzer von KI-Codegeneratoren über kritische Vorfälle und probe-ertesten mit 12 weiteren Entwicklern Vertrauens-Schnittstellen. Vertrauen gründet in wahrgenommener Fähigkeit, Integrität und Wohlwollen des Werkzeugs und ist hochgradig situationsspezifisch; die heutigen Werkzeuge bieten kaum Anhaltspunkte, um Vertrauensurteile zu fundieren (S. 1, S. 5). Für die Moderationsfrage: Das Vertrauensurteil — und damit Annahme oder Prüfung eines Vorschlags — ist eine nutzerseitige Größe mit Qualitätsfolgen, deren Fundierung Werkzeug und Nutzer gemeinsam leisten müssen.

4.4 Randstudien und Protokoll

Zamfirescu-Pereira et al. 2023 („Why Johnny Can’t Prompt”) zeigten, dass Domänenexperten ohne KI-Erfahrung beim Prompt-Design systematisch scheitern — unter anderem, weil sie systematisches Testen vermeiden und ihre Mental Models von LLMs stark von deren Funktionsweise abweichen [nur Abstract; Sekundärzitat der beiden Befunde nach Nguyen et al. 2024, S. 4]. **Wermelinger 2023** testete Copilot an den einfachen Python-Aufgaben eines eigenen Lehrbuchs (8 Aufgaben): Lösungs- und Korrektheitsraten fielen aufgabenspezifisch sehr unterschiedlich aus [nur Abstract]. **Borg et al. 2024** liegen als Registered Report nur in der Stage-1-Form vor (In-Principle-Acceptance, ICSME): zweiphasiges Design, in dem professionelle Entwickler Java-Features mit/ohne KI bauen und die Projekte anschließend ohne KI weiterentwickelt werden, um Wartbarkeit zu messen (S. 1–2). **Es liegen keine Ergebnisdaten vor** — die Studie ist ein Marker dafür, dass die eigentlich passende Versuchsanordnung gerade erst aufgelegt wird.

5. Bewertung gegen die Nullhypothese

Pro Nullhypothese (Funde, die Gleichheit nahelegen)

- Strukturierte Planung und verifikationsorientiertes Prompten unterscheiden sich in der 162er-Survey nicht nach Erfahrungsgruppe (Fawzy et al. 2026, S. 17).
- Kurzfristige Code-Korrektheit und einwöchige Retention ohne KI unterschieden sich zwischen Codex- und Kontrollgruppe nicht signifikant (Kazemitabaar et al. 2023, S. 10).
- Selbst eingeschätzte Codequalität ist über Erfahrungsgruppen hinweg ähnlich (Fawzy et al. 2026, S. 1) — wobei Selbstauskunft nach dem METR-Befund ein schlechter Qualitätsindikator ist.
- Die wahrgenommenen Stärken und Grenzen von GenAI ähneln sich gruppenübergreifend (Prather et al. 2024, S. 2-Kontext; Fawzy et al. 2026, S. 1).

Kontra Nullhypothese (Funde, die Unterschiede belegen)

- Innerhalb der KI-Gruppe lieferten Teilnehmer mit adaptivem Prompt-Verhalten (Parameterschrauben, Umformulieren) häufiger korrekten und sicheren Code (Perry et al. 2023, S. 11) — der direkteste Beleg.
- KI-Generierungsqualität hängt stark von der Tiefe des Prompts ab; Experten filtern Fehler, Novizen können es nicht (Moradi Dakhel et al. 2023, S. 1, S. 25).
- GenAI-Lernen spreizt: Kompetentere profitieren, Schwächere stagnieren mit Scheinkompetenz (Prather et al. 2024, S. 2, S. 21; Kazemitabaar et al. 2023, S. 11).

- Interaktionsmuster mit kognitiver Einbindung erhalten Lernen und Kompetenz unter KI-Nutzung; Delegation untergräbt sie (Shen & Tamkin 2026, S. 1).
- Effektives Prompten erfordert Fähigkeiten, die Anfänger nicht haben (Nguyen et al. 2024, S. 15); Domänenexperten ohne KI-Übung scheitern an Prompt-Design (Zamfirescu-Pereira et al. 2023 [nur Abstract]).
- Das Verständnis selbst ist schlecht kalibriert: erfahrene Entwickler täuschten sich über KI-Vor- und Nachteile um ~40 Prozentpunkte (Becker et al. 2025, S. 2, S. 10).

Evidenzsicherheit (GRADE-Logik, in Worten)

- **Für „KI-Gruppen unterscheiden sich in Codequalität je nach Nutzerverständnis": moderat.** Zwei Experimente (Perry, Kazemitabaar) und ein RCT-Ökosystem (METR) stützen die Richtung; die Moderation selbst wird aber nur explorativ/sekundär berichtet (Perry S. 11), nicht als vorgeregistrierte Primärhypothese.
- **Für „Vermittlung von Verständnis verändert das Ergebnis": niedrig bis moderat.** Interventionsbefund nur bei Liu et al. 2023 (S. 18, Design-Feld), dazu Shen & Tamkin (Interaktionsmuster, S. 1).
- **Für die strenge Nullhypothese (kein Effekt): niedrig.** Die Pro-Befunde betreffen Einzelpraktiken (Planungsprompting) bzw. Kurzzeitanmaße, nicht die Gesamtqualität; sie schließen Moderation über andere Kanäle (Verifikation, Filterung, Kalibrierung) nicht aus.
- **Unabhängige Replikation:** Teilweise (Widening Gap repliziert Prather 2020 unter neuen Bedingungen; Perry und „Johnny Can't Prompt" decken sich in der Mental-Model-Diagnose). Eine direkte Replikation der Moderationsbefunde fehlt.

6. Limitationen des Reviews und der Befundlage

1. **Keine direkte Prüfung:** Keine der Studien randomisiert „vertieftes LLM-Verständnis" und misst anschließend Codequalität. Alle Kausalitätsschlüsse sind indirekt.
2. **Proxies statt Konstrukt:** Prompt-Verhalten, Vorerfahrung, Expertengruppenzugehörigkeit und beobachtete Verifikation messen das Konstrukt nur ausschnitthaft; auch die Item-Validität der Proxies ist ungeklärt.
3. **Kleine Stichproben:** 16 bis 162 Teilnehmer in den Kernstudien; mehrere Autoren benennen die Begrenzung selbst (Prather S. 6; Kazemitabaar S. 14; METR-Fußnoten).
4. **Populationsverzerrung:** Sechs der vierzehn Studien arbeiten mit Studierenden bzw. Lernenden; die Übertragbarkeit auf Berufspraxis ist begrenzt — umgekehrt kommen die Profi-Studien (METR, Wang, Fawzy, Moradi Dakhel) zum selben Vorzeichen.

5. **Outcome-Heterogenität:** Sicherheit (Perry), Korrektheit (Kazemitabaar, Moradi Dakhel), Zeit (METR), Lernen (Shen & Tamkin), Praktiken (Fawzy) — ein gemeinsames Qualitätsmaß fehlt; die Evidenz stützt sich auf die Konvergenz unterschiedlicher Maße.
6. **Feld im Wandel:** Modelle und Werkzeuge altern binnen Monaten (METR nutzt Stand Frühjahr 2025, Shen & Tamkin publizieren 2026); Effektstärken sind zeitgebunden.
7. **Zwei Studien ohne Volltext** (Zamfirescu-Pereira, Wermelinger) sind hier nur auf Abstract- bzw. Sekundärzitat-Niveau berücksichtigt; beide tragen das Kürzel in Abschnitt 4 und 5.
8. **Publikationsverzerrungen sind nicht ausschließbar:** Negativ- und Nullbefunde (etwa zu Produktivität bei Prather: kein Zeitgewinn, S. 21) sind sichtbar, aber eine systematische Grey-Literatur-Recherche (Blog-, Industry-Reports) lag außerhalb des OpenAlex-Mandats.

7. Fazit

Die strenge Nullhypothese — keine Qualitätsunterschiede je nach Verständnistiefe — wird von der vorliegenden, indirekten Evidenz **nicht getragen**. Über vier unabhängige Befundlinien hinweg (Sicherheits-Experiment, Lern-Experimente, Benchmark mit Expertenfilter, Beobachtungsstudien zu Interaktion und Vertrauen) zeigt sich dasselbe Muster: Die Ergebnisqualität des Codes unter KI-Bedingung variiert systematisch mit dem, was der Nutzer über das Werkzeug weiß und wie er es handhabt — Prompt-Tiefe, Verifikationsverhalten, kognitive Einbindung, kalibriertes Vertrauen. Zwei Befunde spitzen das zu: Selbstsicherheit ist ein schlechter Proxy für Verständnis (METR), und die Bewertung von KI-Code ist eine eigenständige Fertigkeit, die mit dem Erzeugen von Prompts nicht automatisch einhergeht (Fawzy). Gleichzeitig bleibt die Evidenz jung, kleinzahlig und indirekt; die direkt passende Versuchsanordnung (Verständnis als randomisierter Faktor, Codequalität als Outcome) steht mit Borg et al. erst als Protokoll bereit. Für Praxis und Lehre folgt aus der Befundkonvergenz bereits jetzt: Wer mit LLMs programmieren lässt, sollte Verifikations- und Prompt-Kompetenz als eigenständigen Lernstoff behandeln — nicht als Nebenprodukt der Werkzeugnutzung.

Literaturverzeichnis (alphabetisch, Vancouver-artig)

1. **Barke S, James MB, Polikarpova N.** Grounded Copilot: How Programmers Interact with Code-Generating Models. Proc. ACM Program. Lang. (PACMPL/OOPSLA1, Article 78) 2023. DOI: 10.1145/3586030. Volltext: ja (arXiv 2206.15000). [Beobachtungsstudie]
2. **Becker J, Rush N, Barnes E, Rein D.** Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. METR Preprint 2025. DOI: 10.48550/arXiv.2507.09089. Volltext: ja. [RCT]

3. **Borg M, Hewett D, Graham D, Couderc N, Söderberg E, Church L, Farley D.** Does Co-Development with AI Assistants Lead to More Maintainable Code? A Registered Report. arXiv 2024. DOI: 10.48550/arXiv.2408.10758. Volltext: ja (Stage-1-Protokoll, keine Ergebnisse).
4. **Fawzy S, et al.** From Prompting to Verification: How Experience Shapes Vibe Coding Practices. arXiv 2026. DOI: 10.48550/arXiv.2605.24521. Volltext: ja. [Survey]
5. **Kazemitabaar M, Chow A, Hsiao CK, Ross J, Ericson B.** Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. ICER 2023:21–30. DOI: 10.1145/3544548.3580919. Volltext: ja (arXiv 2302.07427). [Experiment]
6. **Liu DYY, Sarkar A, Gordon AD.** „What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. CHI 2023. DOI: 10.1145/3544548.3580817. Volltext: ja (arXiv 2304.06597). [Explorativ + Intervention]
7. **Moradi Dakhel A, Kashefi O, Khomh F, Desmarais M, Hemmati H.** GitHub Copilot AI pair programmer: Asset or Liability? J. Syst. Softw. 209:111734 (2023). DOI: 10.1016/j.jss.2023.111734. Volltext: ja (arXiv 2206.15331, Preprint-Fassung). [Benchmark]
8. **Nguyen P-X, Denny P, Nair A, Sarkar A, Becker BA, Leinonen J.** How Beginning Programmers and Code LLMs (Mis)read Each Other. CHI 2024. DOI: 10.1145/3613904.3642706. Volltext: ja (arXiv 2401.15232). [Experiment]
9. **Perry N, Srivastava M, Kumar D, Boneh D.** Do Users Write More Insecure Code with AI Assistants? CCS 2023:2785–2799. DOI: 10.1145/3576915.3623157. Volltext: ja (arXiv 2211.03622). [Experiment]
10. **Prather J, Reeves BN, Denny P, et al.** The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. ICER 2024:469–486. DOI: 10.1145/3632620.3671116. Volltext: ja (arXiv 2405.17739). [Experiment + Interviews]
11. **Shen JH, Tamkin A.** How AI Impacts Skill Formation. arXiv 2026. DOI: 10.48550/arXiv.2601.20245. Volltext: ja. [Experimente]
12. **Wang L, Ma S, Hu X, Ye F, Wang H, Wang S.** Investigating and Designing for Trust in AI-powered Code Generation Tools. FAccT 2024. DOI: 10.1145/3630106.3658984. Volltext: ja (arXiv 2305.11248). [Interviews + Design-Probe]
13. **Wermelinger M.** Using GitHub Copilot to Solve Simple Programming Problems. SIGCSE 2023:172–178. DOI: 10.1145/3545945.3569830. Volltext: **nein** (nur Abstract; ACM- und Repository-PDF durch Zugriffsblockung nicht beschaffbar).
14. **Zamfirescu-Pereira JD, Wong RY, Hartmann B, Yang Q.** Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. CHI 2023:1–21. DOI: 10.1145/3544548.3581388. Volltext: **nein** (nur Abstract; ACM-PDF durch Zugriffsblockung nicht beschaffbar; Kernbefunde nach Sekundärzitat bei Nguyen et al. 2024).

Suchstrategie und Abfragedatum siehe Abschnitt 2 (OpenAlex, 30.9.2026). Sämtliche 14 Kernstudien liegen als Zotero-Items in der Sammlung „Programmieren mit KI" (zotero://select/library/collections/9NP25QCP).

Disclaimer: Dieses Review wurde maschinell von einem KI-Agenten (Qwen Code, LLM: glm-5.3-flash) erstellt; alle Aussagen beruhen auf den genannten Volltexten bzw. sind als Abstract-only gekennzeichnet; es ersetzt keine menschliche Peer-Review.