Fragmented Markers, Mixed Results: A Systematic Review of AI Coding Assistants and Developer Productivity

Annemarie Wittig
Leipzig University
Leipzig, Germany

Alina Mailach
ScaDS.AI Dresden/Leipzig, Leipzig University
Leipzig, Germany

Norbert Siegmund
ScaDS.AI Dresden/Leipzig, Leipzig University
Leipzig, Germany

ABSTRACT

AI coding assistants are increasingly embedded in software engineering practice, yet the literature still offers no cumulative, easily interpretable picture of their effects on software developer productivity. The central problem is not a lack of studies, but a lack of comparability: prior work operationalizes productivity through heterogeneous markers, settings, and reporting styles, making cross-study conclusions difficult. To address this, we conduct a staged systematic literature review. In the first stage, we synthesize studies that explicitly examine developer productivity. In the second, we extend the scope to productivity-related developer outcomes using SPACE as a theory-informed lens, capturing human-centered evidence that explicit productivity studies often omit. Across both stages, we analyze how productivity is operationalized, which confounding factors are discussed, and how reported effects are distributed.

We identify 187 unique productivity and productivity-related markers, while individual studies typically only use five, revealing a broad but highly fragmented evidence base. Explicit productivity research is dominated by time-, output-, and artifact-oriented proxies, whereas the broader SPACE-aligned corpus makes satisfaction, psychological aspects, and well-being substantially more visible. Reported effects are often positive, but this pattern is stronger in perceived and tendency-based findings than in objective or statistically evaluated ones. Overall, the literature suggests that AI coding assistants can support productivity, but current evidence does not yet support simple, context-independent conclusions. The key contribution of this review is therefore a structured basis for making productivity evidence more interpretable, multidimensional, and comparable across studies.

CCS CONCEPTS

• **Software and its engineering**;

KEYWORDS

coding assistants, productivity, systematic literature review

ACM Reference Format:
Annemarie Wittig, Alina Mailach, and Norbert Siegmund. 2026. Fragmented Markers, Mixed Results: A Systematic Review of AI Coding Assistants and Developer Productivity . In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

ASE'26, October 12–16, Munich, DE

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Coding assistants such as ChatGPT and GitHub Copilot are rapidly reshaping software engineering practice across tasks such as code and test generation [4, 20, 43–45, 49], code review [2, 10, 20, 49, 55], and automated program repair [27, 58]. By 2025, 84 % of professional developers reported that they either want to adopt or have already adopted AI coding assistants in their workflow [47], making productivity effects a central practical concern.

Despite the growing number of studies on AI coding assistants, the evidence on developer productivity remains difficult to interpret. Reported effects vary substantially across studies: some find large time savings when completing a task [41], no statistically significant time savings in some phases [7], and some even slower task completion [3]. These differences are not surprising: studies examine different kinds of tasks (researcher-designed exercises [41], realistic maintenance tasks [7], and real repository issues [3]) and use highly heterogeneous productivity measures. Across the literature, "productivity" may refer to task completion time [32], developer perceptions [22, 30, 32, 38], repository activity metrics (e.g., commits, pull requests, lines of code) [18, 51, 59], or observational findings [26, 32]. Even task completion time itself is operationalized in multiple ways (e.g., direct measurement, Likert-scale perceptions, percentage estimates, or qualitative interview results). As a result, findings are hard to compare, and the field still lacks a coherent view of when and how AI coding assistants improve productivity.

This comparability problem is not merely a methodological inconvenience, but directly affects the conclusions that researchers and practitioners draw from the literature. When results from fundamentally different study settings are interpreted side by side as if they were equivalent, the field risks overgeneralizing effects that may depend on task type, population, study environment, or unreported contextual factors. The consequences are twofold: practitioners may base adoption decisions on evidence from study designs that do not reflect realistic software engineering work, and researchers may repeatedly reproduce similar weaknesses because the literature lacks a structured overview of which findings are comparable, under what conditions, and with which confounders in mind. The key question for a review is therefore not only what effects have been reported, but whether existing syntheses make those effects interpretable and comparable in the first place.

Existing literature reviews provide useful orientation, but do not yet make productivity evidence interpretable and comparable

across heterogeneous contexts [15, 21, 25, 35, 54]. Prior reviews either take a broad LLM-in-software-engineering perspective [21, 54], focus on specific populations [15], or summarize effects without systematically tracing operationalizations, study conditions, and contextual factors relevant to comparability [35]. To our knowledge, no existing secondary study jointly analyzes these dimensions to make productivity evidence comparable and interpretable.

We address this gap with a staged systematic literature review (SLR) designed to improve the interpretability and comparability of evidence on AI coding assistants' effects on software developers' productivity. The staged design reflects two analytically distinct synthesis objectives. Explicit productivity studies provide the clearest basis for analyzing how productivity is defined, measured, reported, and under which conditions effects emerge. At the same time, productivity is widely understood as multidimensional, and prior work does not consistently label all developer-centered, productivity-relevant outcomes as productivity. We therefore structure the review in two stages: the first synthesizes explicit productivity studies, while the second synthesizes studies reporting outcomes aligned with one or more dimensions of the SPACE framework (Satisfaction and well-being, Performance, Activity, Communication and collaboration, and Efficiency and flow) — even when productivity is not the primary stated construct. We do not assume prior studies consistently apply SPACE or map cleanly to its dimensions. Instead, we use it as a theory-informed scoping lens, integrating broader human-centered evidence without collapsing into the explicit-productivity corpus. This staged design supports both a precise analysis of direct productivity claims and a broader account of productivity-related effects, reducing construct undercoverage and strengthening the basis for cumulative research and evidence-informed practice.

Our analysis produced a taxonomy that organizes the methodological fragmentation, arranging research insights according to their operationalization approaches. From this taxonomy, we derive three key observations: (i) human-centered dimensions (e.g., developers' well-being) are rarely investigated in explicit productivity research; (ii) confounders are not systematically analyzed; and (iii) opinion- and tendency-based findings are largely positive, whereas statistically evaluated and objective results contain substantially more neutral and negative findings. This strengthens the observation that the central challenge is not the absence of evidence, but the absence of an interpretable and comparable evidence base. We contribute an important structured basis for making heterogeneous productivity findings more comparable and interpretable across software engineering contexts. Our consolidated overview of markers, confounders, and effect directions provides the shared ground the field needs to move from isolated studies toward a cumulative and coherent evidence base. For academia, this supports more cumulative and better-calibrated research designs; for industry, it enables more defensible interpretation of productivity claims when evaluating AI coding assistants for real development environments.

To summarize, our contributions are as follows:

- We conduct a staged systematic literature review of AI coding assistants' impact on software developers' productivity(-related) markers.
- We compile and categorize 187 unique productivity and productivity-related markers, identifying which aspects are measured frequently and which are largely missing.
- We provide a first list of confounding variables analyzed for results in explicit productivity research.
- We synthesize reported effects of coding assistant usage on explicit productivity markers and summarize where evidence is consistent, mixed, or absent across study contexts.
- We derive implications for researchers designing future evaluations and practitioners assessing productivity impact.

2 BACKGROUND AND RELATED WORK

Productivity in Software Engineering. Developer productivity has been studied for decades without converging on a universal definition or metric set. Early approaches relied on output-oriented indicators such as lines of code [24], while recent work recognizes it as multi-faceted and not to be reduced to a single measure [17, 39].

This shift is reflected in multidimensional frameworks such as DevEx and SPACE. DevEx frames developer productivity through developer-centric drivers such as feedback loops, cognitive load, and flow state, and recommends combining subjective perceptions with objective process indicators [39]. SPACE similarly argues for a multidimensional view of productivity and distinguishes five dimensions: Satisfaction and well-being, Performance, Activity, Communication and collaboration, and Efficiency and flow [17]. These frameworks are relevant for two reasons: they motivate why single-metric interpretations of AI coding assistant productivity gains are often incomplete, and they provide a lens for identifying productivity-related outcomes in studies that do not explicitly use the term *productivity* while still reporting developer-centered effects relevant to productivity in practice.

Secondary Studies on AI Coding Assistants and Developer Productivity. Existing review studies provide valuable orientation for this rapidly evolving research area, but do not yet deliver the synthesis required to make productivity evidence interpretable and comparable across heterogeneous contexts [15, 21, 25, 35, 54]. Available reviews differ substantially in scope, population focus, source policy, and synthesis objective, making them informative for mapping the field, but less suitable for resolving the comparability problem that motivates our work.

Ferino et al. [15] focus on novice developers' adoption of LLMs, covering motivations, perceptions, and experiences in only educational and early-career contexts. Mohamed et al. [35] are the closest prior work, conducting an SLR on LLM-assistant productivity across 37 studies, mapping findings to SPACE. Our review differs in analytical purpose: where Mohamed et al. characterize study methods and synthesize benefits, risks, and SPACE-dimension coverage, we explicitly target the *comparability* problem by analyzing marker-level operationalizations, extracting confounding variables, and preserving study-setting differences in effect synthesis. We also use SPACE, but with a different purpose: as a staged scoping mechanism to recover human-centered evidence that an explicit-productivity-only review would miss.

Further, we considered broader LLM-in-software-engineering reviews [21, 54], which map application areas and trends but are not designed to synthesize human-centered productivity evidence

across heterogeneous settings. We additionally examined Friso and Kalava [25], but their scope and reporting style differ substantially from protocol-driven SLRs, limiting direct comparability.

The source policy also varies across prior reviews. Most existing SLRs are restricted to peer-reviewed studies [15, 21, 35], a defensible choice for methodological rigor. However, for productivity impact in practice, influential evidence also appears in industry reports and preprints reporting original study designs and results. Excluding these improves uniformity but risks omitting practically relevant evidence. We therefore include peer-reviewed studies, industry reports, and preprints, but only when they report original empirical designs and results, and we label them explicitly in the analysis.

Taken together, to the best of our knowledge, no existing secondary study is explicitly designed to make AI coding assistant productivity evidence *comparable and interpretable* across studies by jointly analyzing marker-level operationalizations, study settings and documented confounding variables.

3 METHODOLOGY

We conducted a *staged* SLR to synthesize evidence on the impact of AI coding assistants on software development. The core challenge was obtaining interpretable results across heterogeneous study designs, tasks, and measurement approaches, making a statistical meta-analysis of effect sizes currently infeasible. Instead, we aimed for a structured and transparent synthesis of (i) how productivity is operationalized via markers used to assess productivity and productivity-related outcomes in human-centered studies of AI coding assistants, (ii) which confounders are discussed in relation to reported effects, and (iii) how reported effects are distributed across explicit productivity and productivity-related outcomes.

3.1 Research Questions

To support an interpretable synthesis across heterogeneous evidence, we first characterize how prior work operationalizes productivity and productivity-related outcomes, then extract confounding variables, and finally summarize reported effects at the marker level. We state the following research questions:

RQ$_1$. How is the productivity impact of AI coding assistants on software developers *operationalized* and *measured* in existing software engineering research regarding
- **1.1** explicit productivity, and
- **1.2** productivity-related effects based on the SPACE dimensions?

RQ$_2$. Which confounding variables are discussed in relation to the reported explicit productivity effects?

RQ$_3$. What effects of AI coding assistants are reported for explicit productivity markers?

RQ$_1$ focuses on *operationalization* (i.e., which markers are used and how they are measured), RQ$_2$ captures contextual interpretation and limitations (i.e., confounding influences explicitly discussed in the studies), and RQ$_3$ summarizes reported effect directions to enable cautious cross-study comparison under heterogeneous reporting. Essentially, we cannot go directly to RQ$_3$ without a prior analysis and contextualization of how and under which constraints the results were obtained. This is a key and unique aspect of our analysis compared to other secondary studies on this topic.

We use the term *marker* rather than *metric* because many studies are not operationalized via standardized quantitative metrics, but instead with qualitative/mixed-methods, such as interviews. Thus, we define markers as study-reported operationalizations of higher-level constructs (e.g., productivity, satisfaction, activity), including quantitative measures and qualitative operationalizations.

3.2 Review Design and Search Strategy

We design a staged systematic literature review following the SIGSOFT Empirical Standard for Systematic Reviews [42]. Phase one synthesizes studies explicitly examining productivity with AI coding assistants. Phase two extends this with a scoping stage using SPACE to capture productivity-related developer outcomes not labeled as productivity. The full review protocol, including detailed processes and author responsibilities can be found in the supplementary material.

3.2.1 Staged Review Design. We conducted the review in two stages corresponding to RQ$_{1.1}$ and RQ$_{1.2}$ with two complementary synthesis objectives. Phase one focuses on studies explicitly examining developer productivity with AI coding assistants (ICp1), providing the most direct and methodologically interpretable corpus for analyzing how productivity is explicitly operationalized and grounds the phase-specific analyses of explicit productivity markers, confounders, and reported effects. Phase two focuses on productivity-related outcomes aligned with one or more SPACE dimensions (ICp2), acknowledging that relevant developer-centered evidence is not always labeled as productivity.

This design was empirically confirmed during phase one screening: multiple studies reported outcomes conceptually aligned with SPACE dimensions without naming productivity as the primary construct, and would have been excluded under an explicit-productivity-only criterion. Phase one was retained as the explicit-productivity corpus and complemented by phase two as a theory-informed extension, with phase-specific analyses preserved throughout to broaden conceptual coverage without collapsing distinct forms of evidence. Merging the two phases would therefore risk conflating methodologically distinct forms of evidence.

3.2.2 Selection Criteria. We defined inclusion and exclusion criteria to include only studies that provide direct, human-centered evidence relevant to software developers' work with AI coding assistants. The key criteria are summarized as follows:

- **Phase-specific scope criteria:**
  - **ICp1 (Phase one, RQ$_{1.1}$):** The paper explicitly examines productivity effects of coding assistants on software developers.
  - **ICp2 (Phase two, RQ$_{1.2}$):** The paper examines productivity-related effects aligned with one or more SPACE dimensions.
- **Common inclusion criteria (both phases):**
  - Addresses a software engineering activity/developers' work (IC2).
  - Directly reports productivity(-related) results (IC3).
  - Involves human participants/data from human developers (IC4).
  - Publicly available in English (IC5).
  - Peer-reviewed, preprint, or industry report with an original empirical study design and results (IC6).
- **Key exclusions:**
  - Published before 2023, predating the LLM-assistant period (EC1).

– Exclusively teaching/student/non-developer populations (EC2).
– Fully automated benchmark/ground-truth evaluations and LLM–LLM comparisons (EC3).
– Humans only for ground-truth creation/output evaluation (EC4).
– Other gray literature not meeting IC6 (EC5).

Mixed-participant studies (e.g., developers and students) are included when conducted in non-teaching contexts and labeled accordingly. The temporal scope starts in 2023 (EC1), reflecting mainstream LLM availability from 2022 onward. Full criterion formulations are provided in the supplementary material.

3.2.3 Paper Collection and Screening Procedure. To reduce dependence on a single search source and ranking mechanism, we collected candidate papers from three complementary sources.

(a) a venue-based search in major software engineering venues (to cover established software engineering venues),
(b) an automated keyword-based search using the Semantic Scholar API (for scalable retrieval across indexed sources), and
(c) a manual search using Google and Google Scholar (for targeted recovery of additional studies and industry reports).

Venue-based search. We evaluated research track papers of 12 major software engineering venues (including ASE, ICSE, and ESE; full list in supplementary). The venue-based screening covered **3,746** papers in total, resulting in **1** relevant paper in phase one (ICp1) and additional **4** (5 total) relevant papers in phase two (ICp2).

Semantic Scholar API search. As a second source, we queried the Semantic Scholar API using 120 iteratively refined keyphrase combinations and collected up to 20 results per query. Keyphrases were constructed from template strings combining placeholders for person type (PERSON: *human, participant, developer, practitioner, expert*) and tool type (CHAT: *chatgpt, LLM, coding assistant, chatbot, copilot*), combined with fixed keyphrases. Placeholders and combinations were refined iteratively, with unsuitable ones dropped. The template and static phrases are as follows:

- **Templates:**
  – PERSON using CHAT
  – How CHAT impact PERSONs
  – Do CHAT increase productivity of PERSONs?
  – Productivity of CHAT in practice
  – Software PERSON on CHAT-usage
  – Impact of CHAT on Software PERSON
- **Static:**
  – Developers using LLMs
  – LLMs in development
  – how do coding assistants improve productivity in practice
  – Do coding assistants increase productivity of software developers?
  – Developer productivity coding assistants
  – Effectiveness LLMs in Software Engineering
  – Coding Assistants in Software Development
  – Chatbots in Software Development

The full list of template and placeholder combinations is provided in the supplementary material.

After deduplication, this step contributed 893 unique candidate papers, contributing **13** relevant papers in each phase (**26** total).

Manual search. As a third source, we manually queried **10** productive keyphrases in Google and Google Scholar, adapting syntax where needed (e.g., appending `pdf` to reduce blog results, or reformulating *llm impact developers* as *How LLMs impact developers*). Screening the first 20 results per query added 79 further unique candidate papers. One candidate was provided by Google's AI summary. This summary is generated automatically with most Google searches and is thus part of the search interface rather than an active AI tool used in our collection pipeline. This source contributed **13** relevant papers in phase one and **1** in phase two (**14** total).

Screening and full-text verification. We screened candidates by title and abstract against the selection criteria. All screening and verification steps were performed by the first author. The third author independently screened the first 1,000 venue-paper titles as a calibration subset. Disagreements were resolved through discussion. For any undecided paper, the second author was consulted. For abstract screening, the second author reviewed venue paper abstracts for calibration. When abstracts were ambiguous on the *phase-specific scope criterion* (ICp1/ICp2), we inspected the full text for keywords related to (i) the AI tool (e.g., ChatGPT, LLM), (ii) the study population (e.g., human, participant), and (iii) the productivity(-related) construct under consideration. Papers passing abstract screening underwent full-text verification using the selection criteria, with particular attention to participant population and the requirement that productivity(-related) results are reported rather than only intermediate measures. For any paper where the first author remained undecided, the second author was consulted.

We defined a control set of 10 papers identified through a small-scale manual pre-study and used it to validate search completeness. Search expansion was stopped after the third source once all control papers had been recovered.

3.3 Data Processing and Synthesis Procedure

The first two authors aligned on coding decisions after each step; phase-specific differences are stated below.

3.3.1 Step 1: Extracting Markers. In phase one, we extracted all markers used to assess productivity, excluding markers unrelated to productivity (e.g., use-case reporting without productivity implications) even when reported in otherwise relevant papers.

In phase two, we applied the same extraction logic but evaluated whether each paper's objective or RQs addressed developer impact aligned to one or more SPACE-related constructs, rather than explicit productivity. We reassessed the phase-one pool for previously excluded markers that aligned with SPACE, allowing re-inclusion where appropriate. Derived analyses combining existing markers without introducing a new operationalization (e.g., correlations between previously extracted constructs) were excluded.

Operationalizing SPACE alignment (phase two). Because SPACE defines dimensions conceptually without formal classification criteria, we operationalized inclusion by examining each paper's stated objective, introduction, and (where applicable) research questions/hypotheses to determine whether the investigated construct directly aligned with one or more SPACE dimensions from a developer-centered perspective. Mapping required clear conceptual correspondence to a SPACE dimension rather than indirect influence. Cases

with conceptual overlap were resolved through author discussion. For example, Khojah et al. [26] investigate perceived usefulness and trust in a single RQ: usefulness was retained as its operationalization directly addresses SPACE dimensions (e.g., reducing repetitive tasks maps to *Efficiency and Flow*), while trust, a perception *of* the tool rather than a developer-centered outcome, was excluded.

3.3.2 Step 2: Mapping Markers to Normalized Labels. In both phases, we mapped each extracted marker to a normalized *label* via open coding, enabling cross-study comparison across semantically similar but differently described operationalizations (e.g., varying formulations of time savings or code-activity measures). The output of this step is therefore a concept-level normalization layer above the raw study-reported markers. Quantitative markers were generally straightforward to normalize, while qualitative ones required more semantic interpretation. To calibrate the coding scheme, the first two authors jointly categorized an initial subset of 75 markers in a card-sorting session. Then, the first author completed the remainder and discussed ambiguous cases with the second author.

3.3.3 Step 3: Grouping into Higher-Level Categories. We grouped normalized concepts into higher-level categories inductively via open coding (phase one), extending the structure in phase two to accommodate SPACE-aligned markers not present previously. Categories were further grouped into generalizable groups for readability. An overview of all groups, categories, and markers can be found in our results in Figure 1.

3.3.4 Step 4: Collecting explicitly discussed confounding variables. To operationalize RQ_2, we collected confounding variables only when explicitly discussed in relation to extracted explicit productivity markers (phase one). For each marker, we identified variables explicitly linked to the reported effect and assigned them a normalized label, synthesizing *discussed contextual influences* rather than all variables measured in the included studies.

3.3.5 Step 5: Coding Reported Effects. To operationalize RQ_3, we coded the reported effect direction of each phase-one marker using an ordinal scheme: *negative*, *mixed*, *positive*, *neutral/no-effect*. We introduced this harmonization layer because studies report similar concepts using heterogeneous measurement formats (e.g., time measurements, percentage changes, Likert-scale perceptions, or qualitative findings), making a quantitative comparison infeasible.

We proceeded three-ways, depending on reporting format. **(i) Statistical results:** we coded effect direction based on reported significance, with effect sizes extracted if available. **(ii) Proportion-based results:** we derived proportions of positive, neutral, and negative responses and coded the dominant category, using *mixed* when two categories were within 5 percentage points. **(iii) Reported tendencies:** we coded direction based on tendencies stated by the authors. Effect directions were summarized per marker by counting occurrences; multiple results for the same marker within a paper were counted separately when derived from distinct data.

This aggregation inevitably abstracts from deeper analyses in the original studies and compares substantively different result types. Nevertheless, it was necessary given the substantial heterogeneity in operationalizations, analytical depth, and reporting formats across studies. Without a shared comparison layer, the synthesis would risk conflating differences in reporting practice with differences in substantive findings. The directional coding is thus not a meta-analytic effect-size estimate, but a structured and methodologically defensible summary of reported tendencies. We also extracted evidence type per finding to distinguish quantitative metrics, perceptions, and qualitative findings.

4 RESULTS

We analyzed a total of 45 papers (27 in phase one and 18 in phase two), summarized in Table 1. Overall, 26 of the 45 papers were peer reviewed (57.8%), with a lower share in phase one than in phase two (37.0% vs. 88.9%). The most common methods were surveys and experiments, or combinations thereof (*Top Methods*). Participant numbers ranged from 5 to 88022 (*N (min–max)*), depending on the method used. The smallest sample came from a think-aloud study embedded in a larger study setup [4], and the largest from an analysis of participant activity data in repositories [5]. Given this skewed distribution, we report participant ranges rather than averages. Participants were predominantly software engineering professionals. Study duration differed between phases. Phase two papers were almost exclusively one-shot studies (17/18), whereas phase one included a broader spread of durations, including all long-term studies in the sample (6/27). Interestingly, phase one papers used established productivity frameworks only rarely, with 5 papers using SPACE [4, 7, 22, 38, 56] and 3 papers using other frameworks [16, 26, 33].

Table 1: Overview of papers by phase

	Phase One	Phase Two	Total
Unique Paper	27	18	45
Peer-Reviewed	10 (37.04%)	16 (88.89%)	26 (57.78%)
Top Methods	survey (16) experiment (8)	survey (10) experiment (4)	survey (26) experiment (12)
N (min–max)	5–88022	10–268	5–88022
Mixed	2 (7.41%)	3 (16.67%)	5 (11.11%)
Study Length			
one-shot	17 (62.96%)	17 (94.44%)	34 (75.56%)
short-term	4 (14.81%)	1 (5.56%)	5 (11.11%)
long-term	6 (22.22%)	0 (0.00%)	6 (13.33%)

4.1 RQ_1: Markers of productivity research

We report marker results phase-specific. Phase one captures how prior work *explicitly* operationalizes productivity and therefore provides the tighter basis for comparability. Phase two is not intended as a second explicit-productivity corpus. Instead, it focuses on developer-centered outcomes aligned with SPACE dimensions. This approach highlights the aspects that are excluded by an explicit productivity-only lens. We therefore interpret phase differences not simply as differences in marker counts, but as evidence of what explicit productivity research emphasizes versus what it overlooks.

Table 2 summarizes the extracted markers used to assess productivity(-related) impacts of AI coding assistants on developers. Overall, we identified 187 unique markers. Both phases showed the same median of 5 unique markers per paper and a similar category distribution: each contained 19 unique categories, with a median of

Table 2: Overview of markers per paper by phase

	Phase One	Phase Two	Total
Papers with markers	27	31	45
Total Markers	178	240	418
Unique Markers	91	124	187
Unique Categories	19	19	20
Median # unique Markers per Paper	5	5	6
Median # Categories per Paper	3	3	4

3 categories per paper. This suggests a broad corpus-level evidence base but narrow study-level coverage: many markers exist, but each paper captures only a small part of the construct. To test whether the mix of peer-reviewed, industry, and preprint papers affected the identified markers, we used a PERMANOVA. We did not observe statistically detectable differences in marker distributions by peer-review status in either phase one ($p = 0.839$, pseudo-$F = 0.874$, $R^2 = 0.033$), or phase two ($p = 0.107$, pseudo-$F = 1.168$, $R^2 = 0.038$) paper pool, providing no evidence that marker distributions differ systematically by publication type.

Insight

The field has collected a broad productivity evidence base, though the study-level coverage remains narrow. Across the corpus 187 distinct markers are used, but individual studies usually only rely on a small subset, further limiting comparability.

Markers and groups. We identified 187 unique markers, organized into 20 categories and four groups in a tree-level structure. Quantitative evidence was usually easier to consolidate than qualitative evidence: closely related metrics such as "volume of code changes" and "lines of code added" were unified under "#LoC", whereas qualitatively phrased findings about faster work or time savings had to be consolidated at the level of shared meaning. When the same construct appeared across multiple task contexts, we retained a single shared marker to avoid unnecessary fragmentation. The resulting tree is visualized in Figure 1. Only a subset of categories is shown in detail, as displaying every marker would exceed the paper's space. All markers with categories and their corresponding references are provided in the supplementary material.

We identified four main groups of markers: *AI markers*, covering the AI tool itself; *artifact and repository markers*, covering produced code and repository behavior; *organizational and process markers*, covering task, workflow, and organizational outcomes; and *human markers*, covering developers' experiences, perceptions, and well-being. The most frequently addressed categories are *Time* (84 total markers used), followed by *AI assessment* (60) and *Code quality* (43). Because *Time* belongs to the organizational and process marker group, this group is also the largest overall.

Two patterns are particularly notable. First, both phases remain similarly narrow at paper level, as reflected in the identical medians. Second, they differ systematically in *what* they focus on. Phase one contains more artifact and repository markers, whereas phase two is centered more around human and AI markers, and to a lesser extent organizational and process markers. This contrast is also visible in the phase-unique categories: *production activity* (artifact and repository), appears only in phase one, whereas *well-being* (human) appears only in phase two. Phase two therefore does not simply add markers, but captures forms of developer-centered evidence that explicit productivity studies largely leave outside the construct.

At paper level, the most frequent category pairings in phase one are *Direct productivity assessment* with *Time* (9 papers), followed by *Task success and task quality* with *Time*, and *Time* with itself (7 papers each). Self-combinations are common overall, such as *Code quality* and *Repository activity* (5 and 4 papers, respectively). In phase two, the most frequent pairing is *AI assessment* with *Time* (13 papers), followed by self-combinations of *Time* and *AI assessment* (12 and 11 papers, respectively). These pairing patterns highlight that studies often focus on narrow clusters of closely related markers. Such obtained insights are therefore often narrow in scope even when the broader corpus appears rich.

Insight

The phase contrast is a central result. Phase one shows the field's default productivity lens, centered on observable work-performance proxies and phase two shows the human and experiential dimensions that this default lens misses.

Human markers. This group covers the developers' individual experiences, including *Well-being*, *Satisfaction*, *Psychological aspects*, and *Personal career aspects*, with *Satisfaction* being the most frequent category. Of the four main groups, it has the fewest occurrences and is concentrated in the phase two paper pool. Markers related to *Psychological aspects* and *Satisfaction* roughly double or triple in phase two compared to phase one, while *Personal career aspects* and *Well-being* occur almost exclusively in phase two. For instance, *Well-being* is defined by three papers and captured exclusively through perceived markers including stress, physical and mental discomfort, and fatigue. Human markers are therefore where the value of phase two becomes most visible: explicit productivity research includes these outcomes only sparsely, whereas the broader SPACE-aligned set shows that substantial productivity-relevant evidence exists outside explicit productivity labels.

AI markers. This group describes telemetry and metadata on the interactions of developers and tools measured primarily objectively (*AI usage*; e.g., usage frequency or amount of accepted code), as well as developers' perceptions of tool interaction at a more practical level (*AI tooling*; e.g., perceived ease of use and integration challenges). These categories are relatively balanced between the two phases. The most prominent category is *AI assessment*, which captures how developers evaluate the tool itself and can therefore shape both performance and perceptions of productivity (e.g., perceived suggestion accuracy). Overall, *AI assessment* accounts for 68 % (60/87) of all AI markers. While AI markers are more frequent than human markers, they remain the second smallest group overall. Their stronger phase-two presence is driven primarily by the concentration of *AI assessment* markers in that phase.

Artifact and repository markers. This group is the second largest in our paper pool and the only group for which phase one contains more markers than phase two. It contains six distinct categories, although *Production activity* (phase-one exclusive) and *CI activity* contain only three markers each, drawn from one and two papers, respectively. The largest category is *Code quality*, collected in

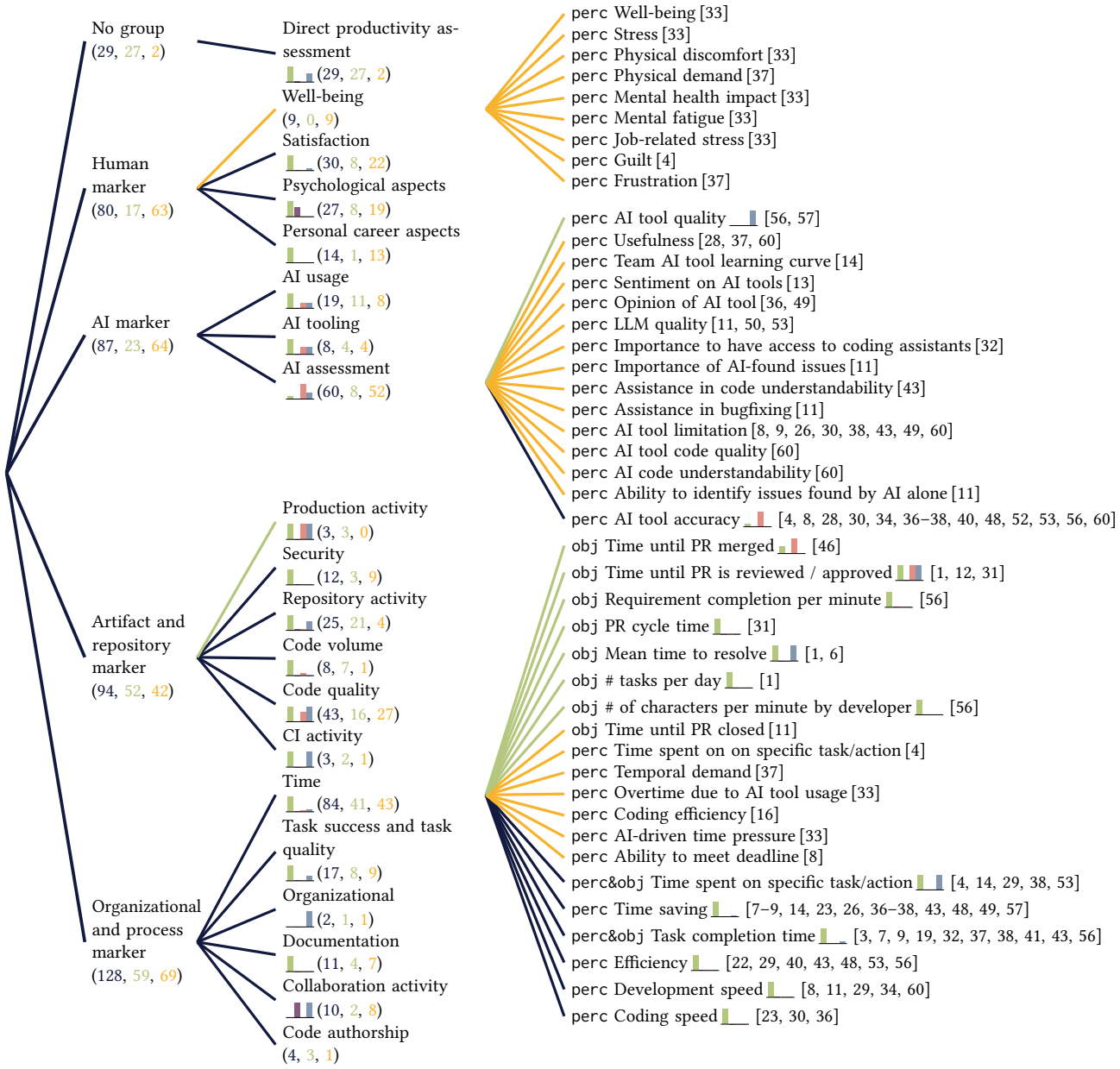

Figure 1: Markers used in productivity research. Green lines indicate phase-one-only markers, categories, and groups; yellow lines indicate phase-two-only; black lines indicate those found in both phases. Numbers below categories and groups show total collected (not unique) markers by phase. Bar graphs show the distribution of positive (green), mixed (purple), negative (red), and neutral/no-effect (gray) results for phase one markers. Leaf labels indicate whether operationalizations are perception-based (`perc`), objective (`obj`), or both (`perc&obj`). Perceived and objective markers are counted separately in the statistics.

both phases but more often in phase two, and spanning 29 unique markers. It includes objective markers such as cyclomatic complexity and test coverage as well as perceived markers such as perceived readability and understandability. A notable subset consists of bug- and issue-related operationalizations and raw counts. Because this group primarily focuses on code and repository artifacts, its markers are predominantly objective, with three categories being entirely objective (*Production activity*, *Repository activity*, and *Code volume*). Specifically, these markers are often operationalized through directly observable outcomes (e.g., #PRs, #commits). The remaining categories are more mixed, but even there many perceived markers are phase-two exclusives. Taken together, artifact and repository markers represent the clearest expression of the explicit-productivity lens foregrounded in phase one.

Organizational and process markers. This is the largest group overall and contains six categories, all present in both phases. The category sizes differ widely, with *Code authorship* and *Organizational* containing only two unique markers each (e.g., amount of AI code in repository and *Company impact*, respectively). *Collaboration activity* captures the social aspects of organizational processes (e.g., satisfaction with communication flows) and occurs primarily in phase two, whereas the other categories are distributed more evenly. Similar to artifact and repository markers, this group is objective-heavy, but some of its most central categories include both objective and perceived operationalizations.

The *Time* category alone accounts for 65 % (84/128) of all organizational and process markers and is distributed evenly across phases, with as many markers based on objective measurements as on participants' perceptions. Two patterns are particularly notable, both of which generalize beyond *Time*. First, some markers are closely related but still not directly comparable. For example, the time until a pull request (PR) is merged is closely related to the time until a PR is reviewed or approved, but not identical, as merging follows approval and may occur later. Second, the operationalization of even central constructs varies strongly across studies, making direct comparison difficult. Task completion time, for example, has been measured directly [3, 7, 9, 37, 41, 56] and reported in seconds, minutes, hours, or percentage changes; it has also been captured through perceived measures, including Likert-scale items [7, 22, 44], other closed-response formats [38, 43], and qualitative coding [19]. These measures were applied at the level of tasks or subtasks, in both artificial and realistic settings, and with different forms of evidence and reporting. As a result, even clearly related findings can usually only be compared at the level of direction, such as whether task completion time improved or worsened, but not by degree.

Insight

The large number of markers should not be read as measurement richness alone. It also signals a fragmented operationalization landscape in which even highly central constructs such as *Time* are often only comparable at the level of direction, not magnitude.

Direct productivity assessment. We define *Direct productivity assessment* as a category of its own because all other markers represent proxy operationalizations, whereas this category captures productivity explicitly from developers through instruments such as Likert-scale items, multi-item questionnaires, and qualitative coding. With 29 total occurrences, it is one of the more common categories. At the same time, these measures vary substantially in wording and scale design. Many Likert-based formulations, not limited to this category, do not permit a clear negative interpretation: disagreement with "My productivity is improved using the AI tool" may indicate either no change or a negative effect. Even where productivity is assessed explicitly, comparability is therefore limited by heterogeneity in how the construct is phrased and elicited.

Insight

A notable phase contrast is that framework-aligned productivity research primarily recovers human-centered outcomes, whereas explicit productivity assessments focus much more strongly on performance and artifact-related proxies. Human aspects therefore appear substantially underrepresented in explicit productivity assessment.

4.2 RQ_2 Confounders of explicit productivity

Across the 27 phase-one papers, we identified 44 unique confounding variables in 15 studies. Of these 44 variables, 21 were reported in a single paper [3], which is also one of the few papers in our sample whose primary finding indicates a negative effect of AI tools on developer productivity. The median number of confounding variables per paper is 1, which stands in sharp contrast to the 44 unique variables identified across the corpus. The most frequently discussed variables are developer experience (6 papers), AI tool experience (4), project/task complexity (4), project involvement (2), usage approach (2), and usage frequency (2).

This pattern suggests that explicit productivity studies do recognize relevant sources of variation, but not in a sufficiently systematic way to support cumulative interpretation across studies. Given the small number of papers focusing on different aspects of productivity, we cannot draw generalizable conclusions about how these variables affect specific markers and therefore report only tendencies observed in the included studies.

Focusing on developer experience, the evidence indicates a trend but no clear consensus. Junior developers appear to benefit more from AI [3, 18, 38, 41, 59], while findings for senior developers are mixed: some studies report smaller gains [18, 38], others find no significant effects [3], and some suggest either exclusive benefits for experienced developers [56] or declines in their productivity [59]. The evidence for AI tool experience and task/project complexity is somewhat more consistent. Prior experience with the tools is associated with neutral, but mostly positive, impacts [7, 22, 51, 56], while increasing task complexity is associated with less positive productivity impacts [22, 38, 53]. Thus, even for the most frequently discussed confounders, the overall picture remains mixed.

The remaining 38 confounding variables were each reported only once, although some overlap conceptually with the top six. For example, programming skills [41] and programming language experience [7] likely relate to developer experience as sub-factors, while others are distinct, such as prompting behavior [56]. Many of these variables are also not specific to the settings of their respective studies and overlap with markers derived in RQ_1. Examples include AI output accuracy [57], AI output speed [57], or AI tool limitations [3], all of which appear in Figure 1. This reinforces that the field has not yet stabilized which constructs should be treated as dependent variables, contextual factors, or confounders.

Insight

Confounding variables are acknowledged, but not standardized. The literature recognizes many potentially important confounders, yet most are discussed only once, which sharply limits cross-study interpretability.

Naturally, this analysis does not imply that these 44 variables are the only confounding variables discussed in the papers, as additional variables may have been collected without being explicitly discussed in relation to productivity findings. Moreover, to avoid blurring the results with unrelated confounders, we excluded the phase-two data from this step and maintain focus on variables discussed alongside explicit productivity findings.

4.3 RQ_3 Explicit productivity effect directions

We include the aggregated effect directions (positive, negative, mixed, or neutral/no effect) of phase-one productivity markers and categories as bar charts in the tree representation in Figure 1. Categories at higher levels aggregate results of lower levels. We chose this aggregation to maintain readability, as many markers occurred only rarely and separate representations for tendency-based, proportional, and statistically evaluated reports would have been overly sparse. Results not clearly positive, neutral, or negative were omitted; for example, accepted code size [56] was reported without directional interpretation by the original authors.

Across explicit productivity markers, positive reported effects dominate overall. However, this aggregated pattern masks a more important result: effect direction varies systematically with reporting format and marker type. Tendency-based findings are decidedly more positive than statistically evaluated findings, and perceived markers are more positive than objective markers. The key result is therefore not simply that positive findings outnumber negative ones, but that the apparent positivity of the literature depends, in part, on how effects are measured and reported.

We report the counts of result directions by reporting type (tendencies, proportions, and statistically evaluated effects) and by marker type (perceived and objective) across all markers in Table 3. Positive results were the most common for all three reporting types. However, the distribution differs substantially across them: tendency-based results show the highest proportion of positive findings, proportional results contain relatively more mixed findings and no negatives, and statistically evaluated effects contain relatively more neutral and negative findings. Likewise, perceived markers show a higher proportion of positive results than objective markers. Overall, about two thirds of all results are positive (63 %), with 10 % negative and 23 % neutral.

Given the small sample size, we used pooled Fisher's exact tests to explore associations between result direction and reporting type (tendencies, proportions, and effects). The omnibus test indicated a significant association ($p < .001$). Post hoc pairwise Fisher's exact tests with Holm correction showed significant differences between all three reporting types (tendencies vs. proportions: adjusted $p < .001$; tendencies vs. effects: adjusted $p < .005$; proportions vs. effects: adjusted $p < .01$). We then applied the same pooled exploratory analysis to perceived versus objective markers and again found a significant association ($p < .05$). By contrast, the association between result direction and gray versus non-gray studies was not statistically significant ($p > .05$), with distributions appearing broadly similar across groups. Because we extracted multiple results from some papers, these pooled contingency analyses do not fully account for within-paper dependence and should be interpreted as exploratory rather than confirmatory.

Table 3: Result direction by reporting type and operationalization; percentages are row-wise.

	positive	negative	neutral	mixed	Total
Tendency	44 (81.5%)	7 (13.0%)	3 (5.6%)	0 (0.0%)	54
Proportional	21 (56.8%)	0 (0.0%)	11 (29.7%)	5 (13.5%)	37
Statistical	41 (53.9%)	10 (13.2%)	24 (31.6%)	1 (1.3%)	76
Perceived	64 (71.1%)	6 (6.7%)	15 (16.7%)	5 (5.6%)	90
Objective	42 (54.5%)	11 (14.3%)	23 (29.9%)	1 (1.3%)	77
Total	106 (63.5%)	17 (10.2%)	38 (22.8%)	6 (3.6%)	167

Insight

The literature is not simply "mostly positive". Positivity is strongest in tendency-based and perceived results, whereas statistically evaluated and objective results contain substantially more neutral and negative findings.

Put differently, the more interpretively lightweight the reporting format, the more positive the reported productivity picture appears. This shifts the interpretation of the overall trend. The central question is not only whether positive findings outnumber negative ones, but whether positivity is robust across stronger and weaker forms of evidence. In our data, the strongest positive skew appears in tendency-based reports, whereas statistically evaluated effects and objective markers show more neutral and negative results. The resulting picture is therefore not one of uniformly positive evidence, but of positivity filtered through heterogeneous reporting practices.

Looking at the distributions shown in Figure 1, most marker categories exhibit predominantly positive reported directions. *Time* is particularly notable, as it is the most frequently reported category and is associated almost exclusively with positive results. Most time-related markers show positive trends, with the main exceptions being the time until a pull request is merged and the time until a pull request is reviewed or approved. One study reported both a significant increase in time until merge (classified negative), yet more timely merges (positive) [46]. For time until review/approval, findings were mixed across studies, with one significant increase, one significant decrease, and one non-significant result [1, 12, 31].

Other largely positive categories include *Direct productivity assessment* and *Repository activity*, albeit with slightly more neutral findings than *Time*. More diverse patterns appear in *Production activity* and *Collaboration activity*, although these categories were reported only rarely (2–3 results each). *Code quality* is overall more positive than negative, but includes a substantial number of neutral results, with 3 of 16 reports classified as negative. Negative findings come from two papers reporting more bugs and issues [1, 46], and one reporting more pull request revisions after the AI tool's introduction [59].

The most negative category is *AI assessment*, with 62 % negative reported results (5/8). These were mostly tendency-based reports concerning AI-tool accuracy, for example open-text responses on hallucinations or incorrect code [40, 56]. This creates a notable tension in the literature: productivity-related outcomes often trend positive, while evaluations of the AI tools themselves are frequently

critical. Developers may therefore perceive practical gains even while remaining skeptical about tool reliability and correctness.

Insight

One of the most interesting tensions in the data is that productivity markers often trend positive, while *AI assessment* trends negative. Developers may experience gains in speed or task completion while still judging the tool itself as inaccurate, brittle, or error-prone.

Similar to the confounding variable analysis, we restricted this step to explicit productivity markers, because comparability was already substantially constrained by the aggregation of heterogeneous reported effects across studies, marker operationalizations, and reporting types. Extending the analysis to the broader marker set would have introduced additional conceptual heterogeneity, further weakening the interpretability of the aggregations.

5 DISCUSSION AND IMPLICATIONS

5.1 RQ_1: Markers of productivity research

The central contribution of this review is not that research on AI coding assistants and developer productivity is scarce, but that the available evidence is difficult to compare in a cumulative way. Across the literature, productivity and productivity-related impacts are operationalized through a broad range of markers, while individual studies usually only capture a narrow subset of them. This indicates that while the field contains numerous relevant findings, there is not yet a coherent evidence base from which more robust conclusions can be drawn easily.

The phase distinction helps interpret what kind of productivity evidence the field currently prioritizes. Phase one shows a default operationalization of productivity that centers on observable work performance, especially time-, artifact-, and repository-related proxies. This is understandable, as such markers are often easier to observe, quantify, and compare within individual study designs. Concurrently, when the broader literature is mapped onto established productivity frameworks such as SPACE, human-centered outcomes, such as satisfaction, psychological aspects, and well-being, occur far more prominently than in phase one. This suggests that current explicit productivity assessment in software engineering tends to underrepresent the human side of productivity, even though these dimensions are central to productivity effects.

This matters because software development productivity is not reducible to output and speed alone. Prior work such as SPACE and DevEx already frames productivity as a multidimensional construct that also includes internal experience, collaboration, and friction in day-to-day work [17, 39]. Our findings suggest that the corresponding conceptual foundation already exists, though practice remains narrower. Therefore, the gap is not one of missing theory, but one of shared research practice. The challenge for the software engineering community is not that every study must measure every productivity dimension at once, but that focused studies should still define and report productivity in ways that make their findings more interpretable relative to one another.

Our taxonomy did not just highlight this gap, it is the first step towards closing it. We grouped markers according to their operationalization rather than by conceptual similarity. This allows researchers to compare two studies not just on the construct, but on how it was measured. They may find whether two studies genuinely disagree or whether their operationalizations were simply incompatible. They may also use the taxonomy to design their studies in a way that makes them inherently better comparable to other related studies with similar markers and goals.

5.2 RQ_2: Confounders of explicit productivity

The findings on confounding variables reinforce the same broader picture. The literature clearly recognizes that productivity effects depend on context, but it does not yet treat that context in a systematic or comparable way. Even the most frequently discussed confounders, such as developer experience, AI tool experience, and task complexity, appear in only a limited number of studies, while many others are mentioned only once. As a result, contextualization exists, but mostly in a fragmented form.

This has two implications. First, the effects of AI coding assistants on productivity are unlikely to be universal. They appear to depend on who is using the tool, potentially the task type and the conditions. The field should therefore move away from asking whether AI coding assistants improve productivity in general, and instead ask under which conditions they help, hinder, or shift work. Second, the overlap between markers and confounding variables indicates conceptual instability. Variables such as AI output accuracy or output speed appear in some studies as outcomes and in others as confounding variables. This makes studies diverge not only in what they measure and how they measure it, but also in how they position the same construct analytically. For cumulative research, that is a substantial problem. For the SE community, this suggests that contextual factors should be treated as part of the core study design rather than optional additions. Not every study needs to cover every possible confounder, but more consistent reporting and justification of key contextual variables would make productivity findings substantially more interpretable across studies.

5.3 RQ_3: Explicit productivity effect directions

At first glance, the literature appears predominantly positive. Across explicit productivity markers, positive reported directions outnumber neutral and negative ones. However, our results suggest that this pattern should not be interpreted as a simple answer. The positivity of the literature is not stable across reporting forms. It is strongest in tendency-based and perceived findings, whereas statistically evaluated and objective findings contain substantially more neutral and negative results. Accordingly, the main interpretation is not that the field has converged on a uniformly positive effect, but that the apparent strength of that effect depends in part on how productivity is measured and reported.

This is important for two reasons. First, some of the most common operationalizations do not allow negative findings to be expressed clearly. For example, positively framed self-report items may collapse negative outcomes into neutral disagreement. Second, perceived gains do not necessarily reflect the same construct as observable performance changes. Future studies should therefore be

more explicit about whether they are reporting perceived productivity, observed behavioral change, or statistically tested effects, and should avoid treating these forms of evidence as interchangeable.

A particularly notable tension in the data is that most outcome categories trend positive, while *AI assessment* appears frequently negative. This suggests that developers can experience tangible improvements in speed, task completion, or output volume, yet simultaneously hold a critical view of the tool itself. For practice, this means that productivity gains and tool quality should not be conflated. A tool may appear beneficial in narrow output-oriented terms while simultaneously introducing correction effort, uncertainty, or maintenance burden. For research, it means that evaluations of AI coding assistants should not focus only on speed or throughput, but also consider such downstream effort introduced by imperfect suggestions. Taken together, the findings from RQ_3 support a more careful interpretation of the current evidence base. The literature points to recurring positive patterns, but not to context- or method-independent conclusions about productivity effects.

6 THREATS TO VALIDITY

Our synthesis involves interpretation-dependent coding decisions. In particular, normalizing markers and confounders required semantic judgment, which may have introduced labeling inconsistencies despite calibration and discussion among authors. We also captured confounding variables only when studies explicitly related them to productivity findings. Therefore, collected but not discussed confounders are absent from the synthesis, but this also reflects inconsistent reporting in the field. In addition, separating explicit productivity studies from other outcomes within the same paper imposed an analytic distinction not always made by original authors, although we mitigated this by revisiting phase-one papers during phase two.

Because the corpus includes peer-reviewed papers, preprints, and industry reports, some aggregated patterns may also reflect variation in rigor and reporting quality across source types. As with any SLR, relevant studies may have been missed despite our multi-source search strategy and control-set validation. Further, applying SPACE in phase two required interpretive judgment in boundary cases.

Our search combined venue-based screening of 3,746 papers across twelve major SE venues, Semantic Scholar API queries, and manual Google Scholar searches, but did not include snowballing or further database coverage, which could increase the paper pool further. However, any resulting gaps in collected papers are unlikely to alter the central finding, given the pronounced heterogeneity and fragmentation of productivity research already evident in the corpus. Finally, many individual markers were reported only in a small number of studies, limiting the strength of directional summaries. Thus, we aggregate mainly at category level and treat statistical analyses as exploratory rather than confirmatory.

7 CONCLUSION

This literature review synthesized how software developer productivity with AI coding assistants is currently studied. Across 45 papers and reports, we identified a broad but fragmented evidence base with many different markers, narrow study-level operationalizations, and limited cross-study comparability. Explicit productivity research focuses primarily on output-, time-, and artifact-oriented proxies, whereas the broader framework-aligned literature makes human-centered dimensions such as satisfaction, psychological aspects, and well-being much more visible. This suggests that current productivity assessment in software engineering captures important parts of the construct, but not yet the full picture.

Our findings further show that reported productivity effects are often positive, but that this positivity depends on how evidence is measured and reported. Perceived and tendency-based results are more positive than objective and statistically evaluated findings, while contextual factors such as developer experience, tool experience, and task complexity are recognized only inconsistently.

Taken together, the current literature does not point to a single stable answer to whether AI coding assistants improve productivity in general. Instead, it points to a field that already contains many valuable findings, but still lacks a sufficiently shared, multidimensional, and context-sensitive research practice. The next key step is therefore not only to produce more productivity studies, but to make them more comparable through clearer specifications of which productivity dimensions are assessed, more consistent reporting and assessing of markers, broader inclusion of human-centered outcomes, and more systematic analysis of confounding variables. If developer productivity with AI coding assistants is a puzzle, the field has already collected many important pieces. This study adds a precise account of which pieces are missing, how they look, and where they fit. The next challenge is to make them fit together.

Data Availability Statement

All data, analysis scripts and additional visualizations can be found at our supplementary material; DOI: https://doi.org/10.5281/zenodo.19219046

Acknowledgements

The authors acknowledge the financial support by the Federal Ministry of Research, Technology and Space of Germany and by Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus in the programme Center of Excellence for AI-research "Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig" (project identification number: ScaDS.AI) and by the European Social Fund (ESF) together with the German state of Saxony under grant number A.100760692 (project: Teaching-AI).

REFERENCES

[1] Faros AI. The ai productivity paradox: Ai coding assistants increase developer output, but not company productivity. Technical report, Faros AI, 2025. URL https://cdn.prod.website-files.com/66f2fa250759eeb1f2b1199a/68d6e93f3060b3c8a32e72bf_AI_Engineering_Impact_Report_July_2025_Faros_AI.pdf.

[2] A. Alami, V. Jensen, and N. Ernst. Accountability in code review: The role of intrinsic drivers and the impact of llms. *ACM Transactions on Software Engineering and Methodology*, 34(8):1–44, 2025. doi: 10.1145/3721127.

[3] J. Becker, N. Rush, E. Barnes, and D. Rein. Measuring the impact of early-2025 ai on experienced open-source developer productivity. arXiv:2507.09089 [cs.AI], 2025. URL https://arxiv.org/abs/2507.09089.

[4] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit. Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools. *Queue*, 20(6):35–57, 2022. doi: 10.1145/3582083.

[5] S. Bonabi, S. Bana, V. Gurbaxani, and T. Nian. Beyond code: The multidimensional impacts of large language models in software development. arXiv:2506.22704 [econ.GN], 2025. URL https://arxiv.org/abs/2506.22704.

[6] J. Bono, J. Grana, K. Karakolios, P. Hanumanthapura Ramakrishna, and A. Srivastava. Security copilot: Evidence of productivity gains in live operations. Technical report, Microsoft Corporation, 2025. URL https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Copilot_productivity_external_Spring2025_042125_v3_remediated.pdf.

[7] M. Borg, D. Hewett, N. Hagatulah, N. Couderc, E. Söderberg, D. Graham, U. Kini, and D. Farley. Echoes of ai: Investigating the downstream effects of ai assistants on software maintainability. arXiv:2507.00788 [cs.SE], 2025. URL https://arxiv.org/abs/2507.00788.

[8] S. Cavalcante, E. Ribeiro, and A. Oran. The impact of ai tools on software development: A case study with github copilot and other ai assistants. In *Proceedings of the International Conference on Enterprise Information Systems*, pages 245–252, 04 2025. doi: 10.5220/0013294700003929.

[9] T. Chen. The impact of ai-pair programmers on code quality and developer satisfaction: Evidence from timi studio. In *Proceedings of the International Conference on Generative Artificial Intelligence and Information Security*, page 201–205. ACM, 2024. ISBN 9798400709562. doi: 10.1145/3665348.3665383.

[10] M. S. S. Chowdhury, M. N. U. R. Chowdhury, F. F. Neha, and A. Haque. Ai-powered code reviews: Leveraging large language models. In *2024 International Conference on Signal Processing and Advance Research in Computing (SPARC)*, volume 1, pages 1–6, 2024. doi: 10.1109/SPARC61891.2024.10829223.

[11] U. Cihan, V. Haratian, A. İçöz, M. K. Gül, Ö. Devran, E. F. Bayendur, B. M. Uçar, and E. Tüzün. Automated code review in practice. In *IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice*, pages 425–436, 2025. doi: 10.1109/ICSE-SEIP66354.2025.00043.

[12] Z. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz. The effects of generative ai on high-skilled work: Evidence from three field experiments with software developers. SSRN Paper, 2025. URL https://ssrn.com/abstract=4945566.

[13] J. K. Das, S. Mondal, and C. K. Roy. Why do developers engage with chatgpt in issue-tracker? investigating usage and reliance on chatgpt-generated code. In *IEEE International Conference on Software Analysis, Evolution and Reengineering*, pages 68–79, 2025. doi: 10.1109/SANER64311.2025.00015.

[14] D. Dhanuka. Impact of llms on team collaboration in software development. arXiv:2510.08612 [cs.SE], 2025. URL https://arxiv.org/abs/2510.08612.

[15] S. Ferino, R. Hoda, J. Grundy, and C. Treude. Novice developers' perspectives on adopting llms for software development: A systematic literature review. arXiv:2503.07556 [cs.SE], 2025. URL https://arxiv.org/abs/2503.07556.

[16] C. Flores-Saviaga, B. V. Hanrahan, K. Imteyaz, S. Clarke, and S. Savage. The impact of generative ai coding assistants on developers who are visually impaired. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–17. ACM, 2025. ISBN 9798400713941. doi: 10.1145/3706598.3714008.

[17] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler. The space of developer productivity: There's more to it than you think. *Queue*, 19(1):20–48, 2021. doi: 10.1145/3454122.3454124.

[18] L. Gambacorta, H. Qiu, S. Shan, and D. M. Rees. Generative ai and labour productivity: a field experiment on coding. Technical Report 1208, Bank for International Settlements, 2024. URL https://www.bis.org/publ/work1208.pdf.

[19] H. Ge and Y. Wu. An empirical study of adoption of chatgpt for bug fixing among professional developers. *Innovation & Technology Advances*, 1(1):21–29, 2023. doi: 10.61187/ita.v1i1.19.

[20] H. Hao, K. A. Hasan, H. Qin, M. Macedo, Y. Tian, S. H. H. Ding, and A. E. Hassan. An empirical study on developers shared conversations with chatgpt in github pull requests and issues. arXiv:2403.10468 [cs.SE], 2024. URL https://arxiv.org/abs/2403.10468.

[21] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang. Large language models for software engineering: A systematic literature review. arXiv:2308.10620 [cs.SE], 2024. URL https://arxiv.org/abs/2308.10620.

[22] B. Houck, T. Lowdermilk, C. Beyer, S. Clarke, and B. Hanrahan. The SPACE of ai: Real-world lessons on ai's impact on developers. arXiv:2508.00178 [cs.HC], 2025. URL https://arxiv.org/abs/2508.00178.

[23] M. Jaworski and D. Piotrkowski. Study of software developers' experience using the github copilot tool in the software development process. arXiv:2301.04991 [cs.SE], 2023. URL https://arxiv.org/abs/2301.04991.

[24] T. C. Jones. Measuring programming quality and productivity. *IBM Systems Journal*, 17(1):39–63, 1978. doi: 10.1147/sj.171.0039.

[25] S. P. Kalava. Ai-powered development: How artificial intelligence is shaping software productivity. *Journal of Artificial Intelligence & Cloud Computing*, 3(2): 1–4, 2024. doi: 10.47363/JAICC/2024(3)E148.

[26] R. Khojah, M. Mohamad, P. Leitner, and F. G. de Oliveira Neto. Beyond code generation: An observational study of chatgpt usage in software engineering practice. *Proceedings of the ACM on Software Engineering*, 1(FSE), 2024. doi: 10.1145/3660788.

[27] J. Kong, X. Xie, M. Cheng, S. Liu, X. Du, and Q. Guo. Contrastrepair: Enhancing conversation-based automated program repair via contrastive test case pairs. *ACM Transactions on Software Engineering and Methodology*, 34(8), 2025. doi: 10.1145/3719345.

[28] P. Kuang, E. Söderberg, and M. Höst. Developers' perspective on today's and tomorrow's programming tool assistance: A survey. In *Companion Proceedings of the International Conference on the Art, Science, and Engineering of Programming*, pages 108–116. ACM, 2024. ISBN 979-8-4007-0634-9. doi: 10.1145/3660829.3660848.

[29] A. Kudriavtseva, N. A. Hotak, and O. Gadyatskaya. My code is less secure with gen ai: Surveying developers' perceptions of the impact of code generation tools on security. In *Proceedings of the Symposium on Applied Computing*, pages 1637–1646. ACM, 2025. ISBN 979-8-4007-0629-5. doi: 10.1145/3672608.3707778.

[30] M. A. Kuhail, S. S. Mathew, A. Khalil, J. Berengueres, and S. J. H. Shah. "will i be replaced?" assessing chatgpt's effect on software development and programmer perceptions of ai tools. *Science of Computer Programming*, 235:103111, 2024. doi: 10.1016/j.scico.2024.103111.

[31] A. Kumar, V. Khare, D. Sharma, S. Kumar, V. Saini, A. Yadav, S. Jain, A. Rana, P. Verma, V. Meena, and A. Edubilli. Intuition to evidence: Measuring ai's true impact on developer productivity. arXiv:2509.19708 [cs.SE], 2025. URL https://arxiv.org/abs/2509.19708.

[32] Gorilla Logic. Ai coding assistants: A gorilla logic experiment. Technical report, Gorilla Logic, 2024. URL https://gorillalogic.com/using-ai-coding-assistants/.

[33] F. N. Meem and B. Johnson. Investigating the impact of ai-assisted tools on software practitioner well-being. In *Adjunct Proceedings of the Annual Symposium on Human-Computer Interaction for Work*, pages 1–5. ACM, 2025. ISBN 979-8-4007-1397-2. doi: 10.1145/3707640.3731915.

[34] W. Mendes, S. Souza, and C. De Souza. "you're on a bicycle with a little motor": Benefits and challenges of using ai code assistants. In *Proceedings of the International Conference on Cooperative and Human Aspects of Software Engineering*, pages 144–152. ACM, 2024. ISBN 979-8-4007-0533-5. doi: 10.1145/3641822.3641882.

[35] A. Mohamed, M. Assi, and M. Guizani. The impact of llm-assistants on software developer productivity: A systematic literature review. arXiv:2507.03156 [cs.SE], 2025. URL https://arxiv.org/abs/2507.03156.

[36] V. Murali, C. Maddila, I. Ahmad, M. Bolin, D. Cheng, N. Ghorbani, R. Fernandez, N. Nagappan, and P. C. Rigby. Ai-assisted code authoring at scale: Fine-tuning, deploying, and mixed methods evaluation. *Proceedings of the ACM on Software Engineering*, 1(FSE):1066–1085, 2024. doi: 10.1145/3643774.

[37] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers. Using an llm to help with code understanding. In *Proceedings of the International Conference on Software Engineering*, pages 1–13. ACM, 2024. ISBN 979-8-4007-0217-4. doi: 10.1145/3597503.3639187.

[38] K. K. B. Ng, L. Fauzi, L. Leow, and J. Ng. Harnessing the potential of gen-ai coding assistants in public sector software development. arXiv:2409.17434 [cs.SE], 2024. URL https://arxiv.org/abs/2409.17434.

[39] A. Noda, M.-A. Storey, N. Forsgren, and M. Greiler. DevEX: What actually drives productivity? *Communications of the ACM*, 66(11):44–49, 2023. doi: 10.1145/3610285.

[40] F. Oliveira, L. Tiago, and L. Chaves. The influence of using llms on the activities of a software testing team: An industry case. In *Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado*, pages 144–146. SBC, 2025. doi: 10.5753/sast.2025.13592.

[41] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. The impact of ai on developer productivity: Evidence from github copilot. arXiv:2302.06590 [cs.SE], 2023. URL https://arxiv.org/abs/2302.06590.

[42] P. Ralph, N. bin Ali, S. Baltes, D. Bianculli, J. Diaz, Y. Dittrich, N. Ernst, Feldererm M., R. Feldt, A. Filieri, B. B. N. de França, C. A. Furia, G. Gay, N. Gold, D. Graziotin, P. He, R. Hoda, N. Juristo, B. Kitchenham, V. Lenarduzzi, J. Martínez, J. Melegati, D. Mendez, T. Menzies, J. Molleri, D. Pfahl, R. Robbes, D. Russo, N. Saarimäki, F. Sarro, D. Taibi, J. Siegmund, D. Spinellis, M. Staron, K. Stol, Storeym M.-A., D. Taibi, D. Tamburri, M. Torchiano, C. Treude, B. Turhan, X. Wang, and S. Vegas. Empirical standards for software engineering research. arXiv:2010.03525 [cs.SE], 2020. URL https://arxiv.org/abs/2010.03525.

[43] D. Russo. Navigating the complexity of generative ai adoption in software engineering. arXiv:2307.06081 [cs.SE], 2024. URL https://arxiv.org/abs/2307.06081.

[44] A. Sergeyuk, Y. Golubev, T. Bryksin, and I. Ahmed. Using ai-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology*, 178:107610, 2025. doi: https://doi.org/10.1016/j.infsof.2024.107610.

[45] J. Shin, C. Tang, T. Mohati, M. Nayebi, S. Wang, and H. Hemmati. Prompt engineering or fine-tuning: An empirical assessment of llms for code, 2025.

[46] F. Song, A. Agarwal, and W. Wen. The impact of generative ai on collaborative open-source software development: Evidence from github copilot. arXiv:2410.02091 [cs.SE], 2025. URL https://arxiv.org/abs/2410.02091.

[47] Stack Overflow. Stack overflow developer survey 2025: AI. Survey report, 2025. URL https://stackoverflow.co.

[48] B. Tabarsi, H. Reichert, A. Limke, S. Kuttal, and T. Barnes. Llms' reshaping of people, processes, products, and society in software development: A comprehensive exploration with early adopters. arXiv:2503.05012 [cs.SE], 2025. URL https://arxiv.org/abs/2503.05012.

[49] M. H. Tanzil, J. Y. Khan, and G. Uddin. Chatgpt incorrectness detection in software reviews. In *Proceedings of the International Conference on Software Engineering*, pages 1–12, 2024. doi: 10.1145/3597503.3639194.

[50] A. Thool and C. Brown. Harnessing the power of llms: Llm summarization for human-centric dast reports. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 33–39. IEEE, 2024. ISBN 979-8-3503-6613-6. doi: 10.1109/VL/HCC60511.2024.00014.

[51] Turing. Using llm coding assistants to increase software development productivity by 33%. Technical report, Turing, 2025. URL https://www.turing.com/resources/llm-coding-assistants-increase-software-development-productivity.

[52] R. Ulfsnes, N. B. Moe, V. Stray, and M. Skarpen. Transforming software development with generative ai: Empirical insights on collaboration and workflow. arXiv:2405.01543 [cs.SE], 2024. URL https://arxiv.org/abs/2405.01543.

[53] T. S. Vaillant, F. D. de Almeida, P. A. M. S. Neto, C. Gao, J. Bosch, and E. S. de Almeida. Developers' perceptions on the impact of chatgpt in software development: A survey. arXiv:2405.12195 [cs.SE], 2024. URL https://arxiv.org/abs/2405.12195.

[54] N. V. Viet and N. T. Vinh. Large language models in software engineering: A systematic review and vision. *Journal of Education For Sustainable Innovation*, 2 (2):146–156, 2024. doi: 10.56916/jesi.v2i2.968.

[55] M. Vijayvergiya, M. Salawa, I. Budiselić, D. Zheng, P. Lamblin, M. Ivanković, J. Carin, M. Lewko, J. Andonov, G. Petrović, D. Tarlow, P. Maniatis, and R. Just. Ai-assisted assessment of coding practices in modern code review. In *Proceedings of the International Conference on AI-Powered Software*, page 85–93. ACM, 2024. ISBN 9798400706851. doi: 10.1145/3664646.3665664.

[56] T. Weber, M. Brandmaier, A. Schmidt, and S. Mayer. Significant productivity gains through programming with large language models. *Proceedings of the ACM on Human-Computer Interaction*, 8(EICS), 2024. doi: 10.1145/3661145.

[57] J. D. Weisz, S. Kumar, M. Muller, K.-E. Browne, A. Goldberg, E. Heintze, and S. Bajpai. Examining the use and impact of an ai code assistant on developer productivity and experience in the enterprise. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. ACM, 2025. ISBN 9798400713958. doi: 10.1145/3706599.3706670.

[58] C. S. Xia, Y. Wei, and L. Zhang. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering*, page 1482–1494. IEEE Press, 2023. ISBN 9781665457019. doi: 10.1109/ICSE48619.2023.00129.

[59] F. Xu, P. K. Medappa, M. M. Tunc, M. Vroegindeweij, and J. C. Fransoo. Ai-assisted programming may decrease the productivity of experienced developers by increasing maintenance burden. arXiv:2510.10165 [econ.GN], 2025.

[60] B. Zhang, P. Liang, X. Zhou, A. Ahmad, and M. Waseem. Practices and challenges of using github copilot: An empirical study. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, volume 2023, page 124–129. KSI Research Inc., 2023. doi: 10.18293/seke2023-077.