Novice Developers' Perspectives on Adopting LLMs for Software Development: A Systematic Literature Review

SAMUEL FERINO, RASHINA HODA, and JOHN GRUNDY, Monash University, Australia
CHRISTOPH TREUDE, Singapore Management University, Singapore

Following the rise of large language models (LLMs), many studies have emerged in recent years focusing on exploring the adoption of LLM-based tools for software development by novice developers: computer science/software engineering students and early-career industry developers with two years or less of professional experience. These studies have sought to understand the perspectives of novice developers on using these tools, a critical aspect of the successful adoption of LLMs in software engineering. To systematically collect and summarise these studies, we conducted a systematic literature review (SLR) following the guidelines by Kitchenham et al. on 80 primary studies published between April 2022 and June 2025 to answer four research questions (RQs). In answering RQ1, we categorised the study motivations and methodological approaches. In RQ2, we identified the software development tasks for which novice developers use LLMs. In RQ3, we categorised the advantages, challenges, and recommendations discussed in the studies. Finally, we discuss the study limitations and future research needs suggested in the primary studies in answering RQ4. Throughout the paper, we also indicate directions for future work and implications for software engineering researchers, educators, and developers. Our research artifacts are publicly available at https://github.com/Samuellucas97/SupplementaryInfoPackage-SLR.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Software and its engineering → Software development techniques**; • **Computing methodologies → Artificial intelligence**.

Additional Key Words and Phrases: Large Language Models, Software Engineering, Novice, Software Developers, Junior, Systematic Literature Review

ACM Reference Format:
Samuel Ferino, Rashina Hoda, John Grundy, and Christoph Treude. 2026. Novice Developers' Perspectives on Adopting LLMs for Software Development: A Systematic Literature Review. 1, 1 (February 2026), 41 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

According to Lenarduzzi et al., *"Software engineering is one of the engineering fields with the highest inflow of junior engineers. The disproportion of junior and senior developers is increasing fast, and it puts a significant stress on the mentoring and tutoring process"*. This is motivated, for example, by the potential cost reduction compared to hiring experienced developers. However, the productivity of junior developers is not at the level of senior developers. LLM tools appear as a candidate alternative to enhance the productivity of novice developers [37], as well as to support senior developers in mentoring junior developers [51]. These tools trained with millions of code lines can support a variety of software development tasks such as code understanding, bug localisation, vulnerability detection, and test case generation [64],

Authors' Contact Information: Samuel Ferino, samuel.demouraferino@monash.edu; Rashina Hoda, rashina.hoda@monash.edu; John Grundy, john.grundy@monash.edu, Monash University, Melbourne, Victoria, Australia; Christoph Treude, Singapore Management University, Singapore, Singapore, ctreude@smu.edu.sg.

with minimal required human effort [160]. Studies are emerging showing the potential advantages related to LLM adoption [37, 45, 161]. Cui et al. [37] conducted a controlled experiment with 4.867 software developers from Microsoft, Accenture, and an anonymous company. They found that the group of developers using GitHub Copilot achieved an increase of 26.08% in the number of completed tasks weekly. Enterprise LLM adoption reports, such as McKinsey [99], Kong [67] and DORA [41], show an increase in companies adopting LLM tools [99], where many companies are exploring the potential benefits (e.g., productivity, job satisfaction) for software development. On the other hand, there is concern that early career software developers might not have mastered the core competencies in software engineering (e.g., coding, and testing) to be able to use those tools properly [73]. Woodruff et al. [150] also mention rising concerns among developers about future implications for their jobs.

In this context, Begel et al. [15] raise an important point about university computer science (CS) / software engineering (SE) graduates becoming novice developers *again* after joining the job market:

> *"Software developers begin a transition from novice to expert at least twice in their careers – once in their first year of university computer science, and second when they start their first industrial job. Novice computer scientists in university learn to program, to design, and to test software. Novices in industry learn to edit, debug, and create code on a deadline while learning to communicate and interact appropriately with a large team of colleagues."* [15]

As soon as they graduate and join a company as junior software developers, they confront the bitter reality of the discrepancy between capstone projects and real-life projects [15, 35]. They also face the difference in expectations while joining the industry [113]. Alboaouh [6] argues that it may take two to three years for a graduate to adjust to industry standards and practices. This includes, for example, improving their technical skills (e.g., learning new frameworks and libraries) by working in large-scale software projects [113]. Thus, early career software developers are also still novice developers in these aspects.

Many studies in Software Engineering (SE) acknowledge the potential effects and implications specific to the population of novice developers and specifically target novice developers as study participants (e.g., [4, 35, 52, 101, 105]). For instance, Zhao et al. [157] investigate the factors (e.g., newbie-manager pairing, job satisfaction) that influence early career software developers. They identified how crucial the early stage is as the foundation for a solid software developer's career, which may incur negative consequences for software developers and development teams if disregarded. In another study, Lenarduzzi et al. [91] identified that junior developers tend to *"incur more code smells over other types of technical debt"*. Because of the profound shift the world has experienced due to the large language model (LLM)-driven revolution sparked by the release of ChatGPT in November 2022 [43, 139], it is also necessary to investigate novice developers' perspectives. In the LLMs for Software Engineering (LLM4SE) research area, Tona et al. [141] detected discrepancies between the perspectives of industry software engineers and SE students through an experimental study. They found that while industry software engineers see the risks LLM tools pose to *"quality of work and professional practice"*, SE students basically perceive those tools as additional tools. That naive mindset regarding the risks of LLM adoption can lead to unforeseeable consequences, as evidenced by one of the participants in Mendes et al.'s study [P: 68]: *"I've had a bad experience because I've uploaded code that I thought was good, but in production, it was unworkable, you know?"*. These situations may compromise the end-user experience (in the worst scenario) as well as contribute to the introduction of bugs.

Systematic literature reviews (SLRs) are emerging in the literature, exploring novice developers adopting LLM tools [26, 115, 117, 119]. However, they focus on understanding how LLM tools are used in computing education

Table 1. State-of-the-art SLRs related to novice developers using LLMs.

Reference	Year	Scope	Target population	Coverage	Time frame	#Included papers
Cambaz et al. [26]	2024	LLMs in CS Education	CS/SE Students	SE Tasks, Perceptions	2018-2023	21
Pirzado et al [115]	2024			SE Tasks	2021-2024	72
Raihan et al. [119]	2024			SE Tasks, Perceptions	2019-2024	125
Prather et al. [117]	2025			SE Tasks, Perceptions	2022-2024	71
Our work	2025	Novice Developers & LLM4SE	CS/SE Students & Junior Developers	SE Tasks, Perceptions	2022-2025	80

[115, 117, 119], such as teaching and learning practices [26] that novice developers utilised LLMs. These SLRs are related to our SLR while investigating novice developers' usage of LLMs, but our SLR differs in terms of purpose and population. These SLRs lack a comprehensive understanding of novice software developers' adoption (e.g., challenges, recommendations) and use of LLMs in SE activities, from the perspective of both CS/SE students and junior software developers. Table 1 summarises the differences between the related works and our SLR.

Seeking to address this research gap, and also motivated by the rise of empirical studies exploring novice software developers' adoption and use of LLMs in SE activities, we conducted a systematic literature review aiming to comprehend novice software developers' perspectives on adopting LLM-based tools for software development tasks. Using the guidelines introduced by Kitchenham et al. [10, 80], we selected 80 primary studies in our SLR from April 2022 to June 2025, identifying the study motivations and goals, methodological approaches, study limitations, study findings, and future research needs. During our analysis, we identified a variety of junior software developers' and CS/SE students' perceptions (e.g., usefulness, emotions, productivity), perceived benefits and challenges, general recommendations, and educational recommendations to support educators in the GenAI era. Although there is no universal understanding regarding junior software developers [141], we employ 2 years of professional experience as a threshold to identify them in the selected studies. Our analysis also covers the software development tasks in which novice software developers are adopting LLM-powered tools. Our work makes the following key contributions:

- Analysis of publication trends covering domains, publication venues, research methods, and LLM-based tools;
- Insights and guidance for IT professionals, educators, and researchers to help them improve their understanding of novice software developers' perception, potentially impacting their decision about LLM adoption;
- A set of key research needs and recommendations for future research directions into Large Language Models for Software Engineering focusing on novice software developers.

This paper is organised in the following sections: Section 2 discusses the background knowledge and key related reviews. Section 3 presents our SLR research methodology. Sections 4, 5, 6, and 7 present the findings for each research question, respectively. Section 8 discusses key study results and suggests directions for future research. Section 9 discusses the study limitations, while section 10 concludes this paper.

2 Background and Related Work

This section provides the background knowledge and discusses key secondary studies in LLM4SE, both in general and focused on novice developers, that are related to our work.

2.1 Definition of Novice Software Developers

As mentioned in the Introduction section, our understanding of novice developers includes both university students and industrial junior software developers. According to Tona et al., there is no universal understanding of the definition of junior developers [141]. This is highlighted by Zhao et al. [157] when they employed five years as a threshold to define early career software developers. In the study by Niva et al. [112], they identified a job description for junior developers that mentioned the requirement of less than three years of professional experience. On the other hand, Gilson et al. [52] use third-year university students as a proxy for junior developers. This demonstrates how flexibly the concept of a junior developer is being addressed by the research community. Tona et al. [141] use the following definition: *"A Junior Software Engineer (Jr.) usually has 0-2 years of experience"*. Note that, since university students *usually* have less than 2 years of professional experience, they would also fit this definition. University students may also have the opportunity to attend industry internships [74, 75, 103]. In our study, we employ 2 years of professional experience as a threshold to avoid ambiguity in this key term. We decided to set a threshold given that we observed during tests with our search string that most of the studies do not characterise participants' expertise, only providing years of experience. Ideally, studies should seek to characterise participants in terms of expertise, combining different mechanisms (e.g., self-assessed expertise) [11].

2.2 Secondary Studies in LLM4SE

Our analysis of the related reviews uncovered seven secondary studies focusing on LLM4SE. Hou et al. [64] conducted an SLR focusing on understanding how LLMs can improve processes and outcomes, selecting 395 primary studies. Similar to our study, their findings revealed the different LLMs employed in software development tasks and strategies used to improve the performance of LLMs in SE. It also includes successful use cases for LLMs. Zheng et al. [159] also conducted an SLR in LLM4SE, selecting 123 primary studies. They discuss the research status of LLMs from the point of view of seven software engineering tasks: code generation, code summarisation, code translation, vulnerability detection, code evaluation, code management, Q&A interaction, and other works. Their findings also present the performance and effectiveness of LLMs in many development tasks. Zhang et al. [155] conducted a systematic survey aiming to summarise the capabilities of LLMs and their effectiveness in SE. Their analysis of the 1,009 selected studies identified what software development tasks are facilitated by LLM tools and factors influencing LLM adoption in SE. Sasaki et al. [125] conducted an SLR of prompt engineering patterns in SE, aiming to organise them into a taxonomy supporting LLM adoption in SE. Based on an analysis of 28 selected studies, their findings resulted in 21 prompt engineering patterns under five main categories. He et al. [59] conducted a systematic review of LLM-Based Multi-Agent Systems for SE, resulting in 71 relevant studies. Their findings include discussing the applications and capabilities of LLM-based multi-agent systems in development (e.g., software maintenance). Fan et al. [46] provide a survey of the emerging areas in LLM4SE, revealing open research challenges regarding the adoption of LLM tools. However, they did not conduct a systematic literature review. We also found a tertiary study by Gormez et al. [53]. They conducted a systematic mapping study unveiling the capabilities and potential of LLMs in software development tasks. They analysed seven systematic literature reviews, identifying LLMs and the challenges faced while using them. Sergeyuk et al. [126] conducted a systematic literature review on human-AI experience in Integrated Development Environment (IDE), identifying 89 primary studies. They also identified benefits and challenges related to the adoption of these tools. To summarise, none of these related secondary studies seek to understand novice software developers' context in using LLMs for Software Engineering.

2.3 Secondary Studies of Novice Developers & LLM4SE

During our search in the literature for secondary studies on novice developers using LLMs for software development, we identified four related works. Cambaz et al. [26] conducted a systematic literature review on the usage of LLM tools for code generation in teaching and learning programming, selecting twenty-one papers published from 2018 to 2023. They searched for terms related to AI tools, computing education, and software engineering in ACM DL, Scopus, and Google Scholar during their paper selection process. They identified educational practices where novice developers utilise LLMs for code generation (e.g., automatic generation of students' assignments). Pirzado et al. [115] conducted a systematic literature review to understand the extent of LLM adoption in computing education. They selected 72 studies from 2021 to 2024, focusing on LLMs used by CS students as coding and debugging assistants. They searched for terms related to AI tools (e.g., Codex), computing education (e.g., Computer Science students), code generation, code explanation, and challenges on IEEE Xplore, ACM DL, ScienceDirect, Web of Science, and SpringerLink during their paper selection process, which was reinforced by a snowballing process. They identified challenges regarding incorporating LLMs in computer education (e.g., lack of accuracy). Raihan et al. [119] selected 125 primary studies from January 2019 to June 2024 during their systematic literature review on the impact of LLMs on computer education. They focused on capturing a general landscape, including the most commonly used programming languages and the LLMs being employed. During their paper selection, they searched for terms related to software engineering (e.g., software development), education (e.g., teaching), and LLMs on ACM DL, IEEE Xplore, Science Direct, Scopus, ACL Anthology[1], Web of Science, and arXiv. They identified students' and instructors' sentiment regarding using LLMs in computer science education as mostly positive in the studies. Prather et al. [117] selected 71 primary studies until May 2024 during their systematic literature review on how instructors integrate generative AI into computing classrooms. They identified a few software development tasks that CS students are using LLM tools (e.g., writing code). Their paper selection process involved searching for terms related to computing education (e.g., computer science education), LLM tools (e.g., ChatGPT), education (e.g., pedagogy), and research methods (e.g., interview) on ACM DL, IEEE Xplore, Scopus, ASEE Peer[2], and arXiv. They identified CS students using LLMs for a few SE activities such as debugging, code generation, and code review.

In summary, previous secondary studies focused on understanding the adoption of LLM tools by novice developers in an educational context. None of these related secondary studies seeks to understand novice software developers' context in using LLMs for Software Engineering, combining both CS/SE students and early career industry developers. Although secondary research focused on novice developers using LLMs is growing, there is no SLR that summarises those novice developers' perceptions and the software development tasks they are using LLMs. Our SLR has a key focus on novice developers' perspectives, including analyses of empirical studies with CS/SE students and early career industry developers. Table 1 summarises the differences between the related works and our SLR.

3 Methodology

Figure 1 shows an overview of the SLR research methodology. Initially, the first author developed a preliminary SLR protocol, which was polished through many discussions with PhD supervisors and other SLR experts. The first author used Parsifal[3] - a platform focused on supporting SLR studies - to support the SLR study design and execution. We also used spreadsheets for the snowballing process, since Parsifal does not support snowballing, and for data extraction and

[1]https://aclanthology.org
[2]https://peer.asee.org
[3]https://parsif.al

analysis. The first author applied the search string to the seven databases, filtered the relevant studies, and extracted and analysed the data by synthesising tables and figures. Each step conducted by the first author was done under the guidance of his PhD supervisors. We detail the research methodology in the following subsections. The research artifacts (data synthesis spreadsheet, data extraction form, SLR Protocol) are available here [48].

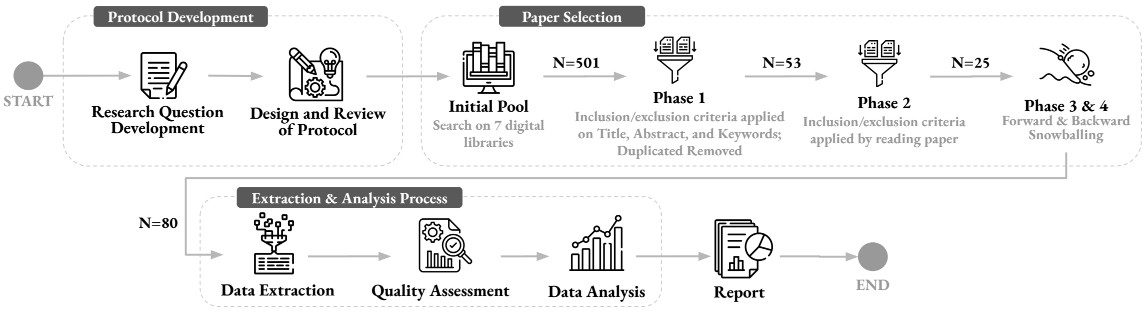

Protocol Development
START
Research Question Development
Design and Review of Protocol
Paper Selection
Initial Pool
Search on 7 digital libraries
N=501
Phase 1
Inclusion/exclusion criteria applied on Title, Abstract, and Keywords; Duplicated Removed
N=53
Phase 2
Inclusion/exclusion criteria applied by reading paper
N=25
Phase 3 & 4
Forward & Backward Snowballing
N=80
Extraction & Analysis Process
Data Extraction
Quality Assessment
Data Analysis
Report
END

Fig. 1. Systematic Literature Review process applied based on Kitchenham et al. [10, 80].

3.1 Research Questions

To develop our research questions (RQs), we employed the PICOC framework (population, intervention, comparison, outcomes, and context), described in Table 2. Wohlin et al. [149] suggested the adoption of the PICOC framework as a foundation to develop well-defined research questions for SLRs. Other research questions were added as a result of discussions between the first author and his PhD supervisors. Thus, our study proposes to address the following research questions:

Table 2. PICOC for Research Questions

Population	Novice software developers
Intervention	Large Language Models (LLMs)
Comparison	N/A
Outcome	Novice software developers' perceptions, challenges and recommendations regarding effects on LLM adoption for software development
Context	Software Engineering

- **RQ1**. *What are the **motivations** and **methodological approaches** behind each primary study to explore how novice software developers adopt LLM-based tools for software development tasks?* – This RQ examines the primary goals, objectives, motivations, and methodologies employed by researchers to identify human aspects of novice software developers who adopt LLM-based tools for software development tasks. For example, we examine whether the study was conducted in an academic or industry setting. We also examine their strategies to categorise less experienced developers, i.e. novice software developers.
- **RQ2**. ***What key software development tasks novice** developers are using LLM-based tools for?* – In this RQ, we examine the software development tasks (e.g., coding, debugging, tests) that novice developers have been supported in by LLM-based tools. We also identify which LLM-based tools are being used by the novice software developers/team.

- **RQ3**. *What are the **perceptions** and **experiences** of novice software developers when using LLM-based tools?*
  RQ3a. *What are the perceived and experienced **advantages/opportunities** of novice software developers when using LLM-based tools?*
  RQ3b. *What are the perceived and experienced **challenges/limitations** faced by novice software developers while using LLM-based tools?*
  RQ3c. *What are the **recommendations/best practices** suggested by novice software developers while using LLM-based tools?*
  In this RQ, we examine the novice software developers' perceptions involving benefits, challenges, limitations, and recommendations based on their experience using those LLM-based tools.
- **RQ4**. *What are the **limitations** and **recommendations for future research** that we can distil based on the primary studies?* – In this RQ, we analyse the studies in terms of key contributions, limitations (e.g., in evaluation, participants), and future research recommended by the authors. Based on this, we suggest future work areas.

3.2 Search Strategy

3.2.1 Search String. We built our base search string from the PICOC framework (See Table 2). We included relevant LLM-based tools such as ChatGPT and Copilot, and terms related to software practitioner roles. Based on the common and relevant database sources used in software engineering literature reviews (e.g., [44, 123]), we decided to include the following six well-known database sources: ACM Digital Library, IEEE Xplore, SpringerLink, Scopus, ScienceDirect, and Wiley. We also included arXiv because research in LLM4SE is an emerging topic, and potentially relevant studies would be under review. Our search was limited to papers published from 2022 – the year that ChatGPT became available for the general public – to July 2025. We refined our base search string following the requirements and setups described by each database. The refinement process was performed several times by applying the search string to the data sources and reading the titles, abstracts, and keywords of some papers. It was necessary to create smaller combinations of our search string for the ScienceDirect database due to word limitations. The research artifacts contain the final search strings for each database. Below, we present the base string:

- *("LLM" OR "large language model*" OR "ChatGPT" OR "Copilot" OR "Generative AI" OR "Conversational AI" OR "Chatbot*") AND (("junior*" OR "novice*") AND ("software developer*" OR "software engineer*" OR "software practitioner*" OR "programmer*" OR "developer*"))*

3.2.2 Inclusion and Exclusion Criteria. During the preparation of this SLR, we defined the inclusion and exclusion criteria. Table 3 shows the inclusion and exclusion criteria adopted during paper selection.

3.3 Paper Selection

The selection was structured using the following steps:

- **Initial Pool:** The search strings were executed across the seven data sources in August (2024), retrieving 501 BibTex references. The BibTeX references containing title, abstract, and keywords were uploaded to the Parsifal platform.
- **Phase 1:** Then, the papers were filtered by their title, abstract, and keywords, while applying the inclusion and exclusion criteria. We removed 40 duplicated papers using Parsifal's duplicate papers detection feature. We also decided to keep the papers that we found difficult to decide based only on their titles, abstracts, and keywords. At the end of this phase, 53 papers were chosen.

Table 3. Inclusion and Exclusion criteria

ID	Inclusion criteria
I01	The paper is about novice developers using LLM-based tools, including junior developers (0-2 years of industry experience) and CS/SE students
I02	The paper must answer at least one of the RQs
I03	The paper is written in English
I04	The full-text is accessible
I05	The paper is not a duplication of others
I06	The paper was published in journals, conferences, and workshops
I07	The paper is an empirical study
I08	The paper was published from 2022, when ChatGPT became available for public access
ID	**Exclusion criteria**
E01	Short papers that are less than four pages
E02	Papers based only on authors' personal views without supporting data
E03	Conference or workshop papers if an extended journal version of the same paper exists
E04	Non-primary studies (Secondary or Tertiary Studies)
E05	Papers about educational contexts not including Computer Science and SE students' perceptions about using LLM-based tools

- **Phase 2:** The papers were filtered by reading in full while applying the inclusion & exclusion criteria. It resulted in 25 primary studies. Table 4 shows details regarding paper count according to each data source. This phase was concluded in September (2024).
- **Phase 3 (Snowballing - Round 1):** Manual search was employed using both *backward* and *forward snowballing* techniques over the 25 primary studies in October (2024). It was utilised Google Scholar during this phase. According to Wohlin [148], snowballing in SLRs is a key technique to complement the automated search on databases, reducing the risk of relevant studies not being included in the paper selection. The snowballing process was organised into four sub-phases (3.1, 3.2, 3.3, and 3.4). For sub-phase 3.1, the papers were filtered by title. For sub-phase 3.2, the papers were filtered based on the abstract and keywords. For sub-phase 3.3, the papers were filtered by skimming the introduction, methodology, results, and conclusion sections. In sub-phase 3.4, we read in full. At the end of this phase, 31 papers were selected, adding to the collection of primary studies. Table 4 presents the paper count for each sub-phase of the snowballing process.
- **Phase 4 (Snowballing - Round 2):** Given the fast-paced research area, we conducted a second round of forwarding snowballing in June (2025) over the 56 primary studies selected in previous phases. We conducted phase 4 similarly to phase 3 (e.g., skimming title, abstract and keywords). It resulted in an additional 33 papers.
- **Phase 5:** We revised the 89 selected papers to ensure that all papers, including industry junior software developers, follow the definition - professionals between 0-2 years of experience. That was necessary because we started the paper selection process using 0-5 years of professional experience as the threshold to categorise junior developers. During this process, we removed nine papers that we previously categorised as junior developers category under 0-5 years of experience. Thus, the paper selection process results in 80 primary studies.

We obtained 80 primary studies from our paper selection process. The ACM digital library was the database source from which we retrieved the highest number of relevant papers - 16.2% (n = 13). At the same time, the snowballing forwarding approach exceeds the ACM DL by returning the highest number of relevant papers - 66.2% (n = 53). The majority of the primary studies are conference papers (67.1%, n=47). Our selected papers also include ten papers published on arXiv. Figure 2a shows the selected papers, including papers from 2022 to 2025. From 2022 to 2024, there

Table 4. Breakdown of the paper count.

	Resource	Initial Pool	Phase 1	Phase 2
PRIMARY SEARCH	ACM DL	184	23	13
	IEEE Xplore	9	3	2
	Springer	70	5	2
	Wiley	27	1	0
	Scopus	51	8	5
	ScienceDirect	107	7	1
	arXiv	53	6	2
	COUNT	501	53	**25**

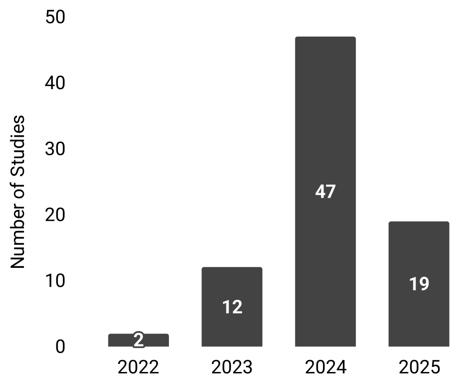
	Resource	Initial Pool	Phase 3.1	Phase 3.2	Phase 3.3	Phase 3.4
SECONDARY SEARCH	Backward Snowballing	1645	67	18	12	11
	Forward Snowballing	924	75	41	27	20
	COUNT	2569	142	59	39	**31**
	Resource	**Initial Pool**	**Phase 4.1**	**Phase 4.2**	**Phase 4.3**	**Phase 4.4**
	Forward Snowballing	3963	136	106	68	**33**

TOTAL FINAL PAPER COUNT (Primary Search + Secondary Search - Phase 5)	**80**

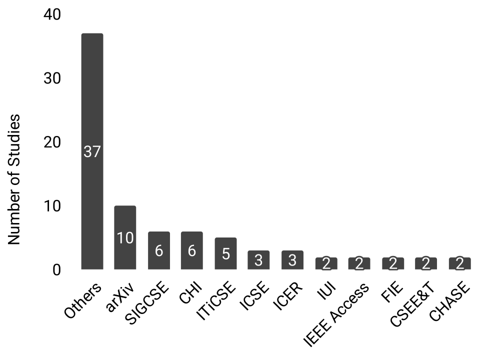
is a consistent trend in the number of publications by year. It is already past the halfway point of 2025 (the current year), and the number of publications in 2025 has already surpassed those in 2023.

Figure 2b shows the distribution of the publication venues, highlighting the venues with more than one publication. In terms of venues, 58.7% of the studies were published in a CORE-ranked conference (A*, A, B, Australasian B, C, or National: Romania) or a ranked journal (Q1 or Q2). The Conference on Human Factors in Computing Systems (CHI) - the most prestigious conference in the field of Human-Computer Interaction - and the Technical Symposium on Computer Science Education (SIGCSE). For journal publications, we identify a few publications in HCI-focused journals such as ACM Transactions on Computer-Human Interaction (TOCHI) and Proceedings of the ACM on Human-Computer Interaction, but there is also a publication in the ACM Transactions on Software Engineering and Methodology (TOSEM) and another in the Information and Software Technology (IST), general-purpose software engineering journals.

Figure 3 presents the distribution of publication domains over the years, based on analysis of paper title, abstract, and keywords. Computing Education and Human-AI Interaction are the two most relevant domains, encompassing 88.7% (n = 71) of the selected primary studies. We perceive a trend in publications focused on those two domains. However, a diversity of domains is emerging in 2024 (e.g., Game Development, Virtual Reality Development).



(a) Distribution of the studies over years.



(b) Distribution of the studies across venues.

Fig. 2. Overview of the 80 primary studies.

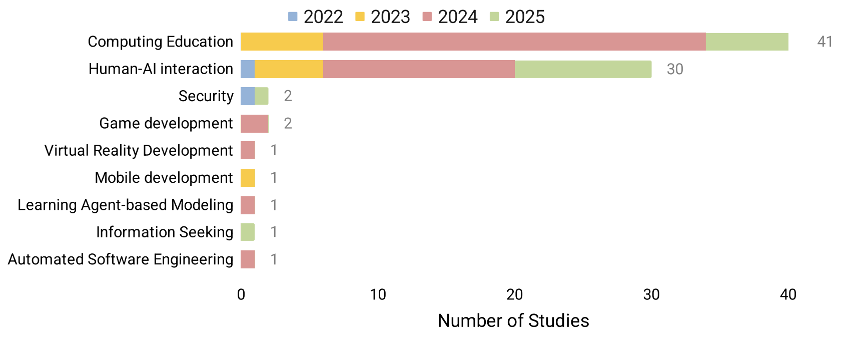


Fig. 3. Distribution of domains over years.

3.4 Data Extraction and Analysis

The information necessary to address the RQs was extracted by inserting the data from each paper into a Google form composed of 40 questions, organised in the following 5 sections:

(1) general information (e.g., paper title, paper's authors, year, venue);
(2) motivations and methodological approaches (e.g., study goal, research question);
(3) key software development tasks (e.g., key software development tasks, LLM-based tools being used);
(4) perceptions about LLM4SE (e.g., benefits, challenges);
(5) limitations and future research needs (e.g., main findings and limitations).

This final version of the data extraction form was the result of adjustments after a pilot test with 21 papers. Our definition of novice developers encompasses both university students and industry junior software developers (0-2 years). The novice developers' perceptions in the studies with other study participant roles (e.g., senior developers, instructors) were collected by cross-checking with participant demographics described in the paper or supplementary material. For instance, Russo's study [P: 8] comprehends novice and experienced developers as study participants; however, we only extracted the novice developers' perspectives. The data extracted from the papers is available in the SLR data (supplementary package [48]), in the Form Responses sheet.

The first author conducted a descriptive statistical analysis of the data using graphs and tables under the supervision of the other authors. Categories for the study motivations were created by first summarising the data extracted from the papers, allowing familiarisation with similarities in the study goals and motivations. Then they were grouped by categories and sub-categories. We did something similar during the categorisation of the benefits, challenges, recommendations, study limitations, and study recommendations for future work.

3.5 Paper Assessment

Seeking to facilitate the understanding of the structure of our selected studies, we performed a paper assessment based on a yes-no-partial evaluation system. Our paper assessment strategy follows the literature on software engineering [61, 81]. Table 5 shows our paper quality assessment strategy based on eight predefined questions. Similarly to Khalajzadeh and Grundy [78], we employed the Computing Research and Education Association of Australasia (CORE[4]) Conference and the Scimago Journal Rankings[5] to identify reputable venues (QA8). The results of the assessment are available in the

[4]https://portal.core.edu.au/conf-ranks
[5]https://www.scimagojr.com

Table 5. Paper quality checklist criteria.

ID	Criterion
QA1	Is the paper highly relevant to the proposed SLR?
QA2	Is there a clear statement of the aim of the research?
QA3	Is there a review of key past work?
QA4	Is there a clear research methodology which aligns with the key research questions of the study?
QA5	Does the paper provide sufficient information on data collection and data analysis of the research?
QA6	Are the findings of the research clearly stated and supported by the research questions?
QA7	Does the paper provide limitations, summary and future work of the research?
QA8	Is the paper published in a reputable venue?

supplementary package. Note that arXiv papers (not published) will receive *No* in QA8. Papers published in conferences not ranked in CORE but have clear sponsorship by ACM or IEEE, traditional computing organisations, will receive *Partially*, such as IEEE/ACM International Conference on Cooperative and Human Aspects of Software Engineering (CHASE). We decided to follow some SLRs (e.g., [60, 79, 109, 123]) in not excluding papers based on quality assessment.

4 RQ1: What are the motivations and methodological approaches behind each primary study to explore how novice software developers adopt LLM-based tools for software development tasks?

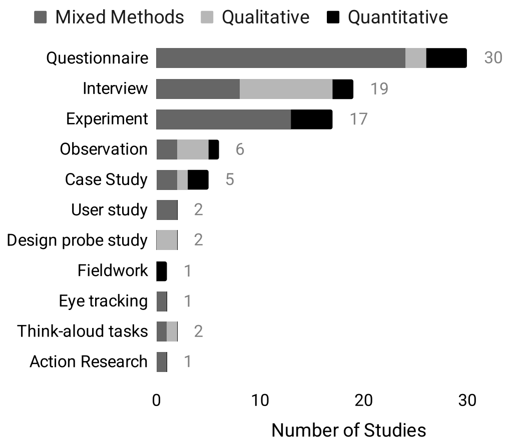
This section contains the findings regarding the first research question. We first present the study motivations, goals, and objectives identified in the primary studies, followed by the study methodologies and data analysis techniques.

4.1 Study motivations, goals, and objectives

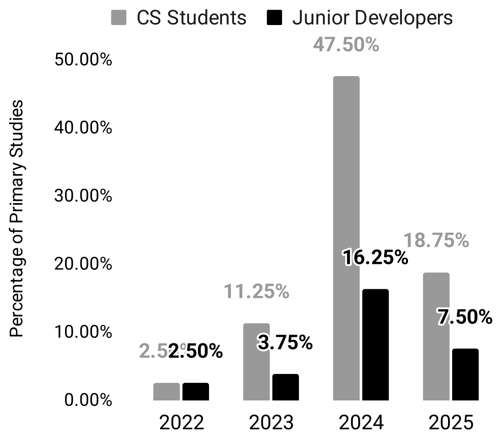
Through our analysis of the motivations, goals, and objectives presented in each of the 80 primary studies, we classified them into four categories: integrating LLMs in SE and its implications, integrating LLMs in SE Education, with its implications, and integrating LLMs in specific industry domains. We identified 29 primary studies that explored novice developers' perceptions and attitudes (e.g., trust) towards LLMs, as well as their effectiveness in improving the productivity of software engineering tasks. For instance, Yang et al. [P: 36] sought to understand the impact of LLM tools during debugging. We found 51 primary studies focusing on the integration of LLMs into Computer Education, including aspects such as perceptions, attitudes, advantages, and challenges. For instance, Shah et al. [P: 50] seek to understand how CS students use GitHub Copilot. We found two primary studies investigating the adoption in specific industries. For instance, Boucher et al. [P: 14] seek to understand the impact of LLM adoption on game development. In this sense, we suggest that future research explore other domains, especially those that impose restrictions on privacy, such as government and finance.

4.2 Study methodologies and data analysis techniques

Figure 4a shows the distribution of the research methods in the 80 primary studies. Questionnaires (30 studies, 37.5%) and interviews (19 studies, 23.7%) were the most recurrent data collection methods. The interviews commonly followed the semi-structured format, but there is also Perry et al.'s study [P: 20] and Choudhuri et al. [P: 62] where they employed retrospective interviews and reflective interviews, respectively. Most of the selected studies (49, 61.3%) used mixed data analysis methods, while 18 studies (22.5%) used qualitative methods, and 13 studies (16.3%) used quantitative methods. While analysing the Figure 4a, we noticed the potential of researchers to explore the implications of novice developers adopting LLM tools using the less commonly used research methods. For instance, researchers could replicate the think aloud-based study by Salerno et al. [122], which explored challenges faced by novice developers when installing tools, in the context of LLM tools. Researchers could also replicate the study by Silva et al. [132], which investigated novice developers' code comprehension using eye tracking. Thematic analysis (36 studies, 48.2%) and grounded theory (5 studies, 6.2%) are the most commonly used qualitative data analysis techniques. Regarding quantitative data analysis techniques, we identified a variety of 33 techniques (e.g., student t-test, Mann-Whitney U test). Descriptive analysis was the most commonly employed quantitative data analysis technique - 33 studies, 41.2%. Figure 4b shows the distribution of study participants: CS/SE students and industrial junior software developers. Most of the studies investigate novice developers by using university students. The literature lacks studies in industry settings that are more realistic.



(a) Distribution of research methods.



(b) Distribution of the study participants.

Fig. 4. Overview of methodologies across the 80 primary studies.

RQ1. What are the motivations and methodological approaches behind each primary study to explore how novice software developers adopt LLM-based tools for software development tasks?

We classified the goal stated in the primary studies into four main categories: integrating LLMs in SE (36.2%), integrating LLMs in CS Education (63.7%), and integrating LLM tools in specific industry domains (2%). Questionnaires, interviews, and experiments were the most common data collection techniques, and both qualitative and quantitative methods were used for data analysis. Most studies have been done with academia-based projects.

5 RQ2: What key software development tasks are novice developers using LLM-based tools for?

This section contains the findings regarding the second research question. While we focus on novice developers, we provide an analysis of the use of LLMs in different SE activities similar to Hou et al. [64], according to the SE activities in the Software Development Life Cycle (SDLC). The SDLC is organised in the following SE activities: requirement engineering, software design, software development, software quality assurance, software maintenance, and software

Table 6. SE Activities in which LLM tools are employed by novice developers.

SE Activity	SE Tasks	
Requirement Engineering & Software Design	brainstorming (20)	problem understanding (14)
	project requirement (2)	to create storyboards (1)
	diagram generation (2)	visualizing user scenarios (1)
	architecture definition (1)	user interview question generation (1)
	GUI mockups (1)	design a framework template (1)
	use case generation (1)	
Software Development & Software Quality Assurance	information retrieval (55)	code generation (45)
	conceptual understanding (32)	code understanding (28)
	documentation generation (9)	to correcting syntax (8)
	unit test generation (6)	test case generation (5)
	to generate regular expression (1)	data test generation (2)
	game content generation (1)	image generation (1)
	generating fake data for prototypes (1)	code comment generation (1)
	regression testing generation (1)	validate JSON (1)
	commit message generation (1)	code translation (1)
Software Maintenance	debugging (49)	code refactoring (10)
	code review (7)	code analysis (4)
	rubber duck debugging (2)	data analysis (1)

management. They can be sequential or iterative depending on the SE methodology; however, the activities are more or less the same. Table 6 shows the distribution of SE tasks across 75 selected papers, including the number of paper occurrences. We did not identify reports of novice developers using LLMs for software project management tasks. According to the literature in SE, project management includes tasks such as effort estimation [25, 70, 84, 106] and task prioritisation [65, 153]. Kula et al. [84] and Molokken et al. highlighted that task effort estimation is heavily based on experts' past experiences. Since novice developers lack prior experience, they can be overly optimistic when estimating task effort [69]. A similar situation happens in task prioritisation [65]. Based on this, there is a research gap involving investigations on the potential of LLM tools to support novice developers in effort estimation and task prioritisation.

Figure 5 shows the distribution of 29 LLM tools in 73 studies. We found LLM tools designed to assist general activities (e.g., ChatGPT, Claude) and development (e.g., GitHub Copilot, Phind[6]). We also found LLMs focused on image content generation (i.e., MidJourney and Dall-E). ChatGPT has been the most recurrent LLM tool in studies since 2023. We also identified an increase in the variety of LLM tools since 2024. Although all 73 studies mention proprietary LLMs, the open-source DeepSeek [42, 57] appeared for the first time in a publication in 2025. Whereas it is an arduous task to find alternative open-source LLMs that can challenge famous closed-source LLMs like ChatGPT and GitHub Copilot, Yang et al. [154] explored and categorised the ecosystem of LLMs for coding tasks available in Hugging Face - the premier hub for transformer-based models. At the same time, we observe a gap in the literature regarding investigations

[6]https://www.phind.com

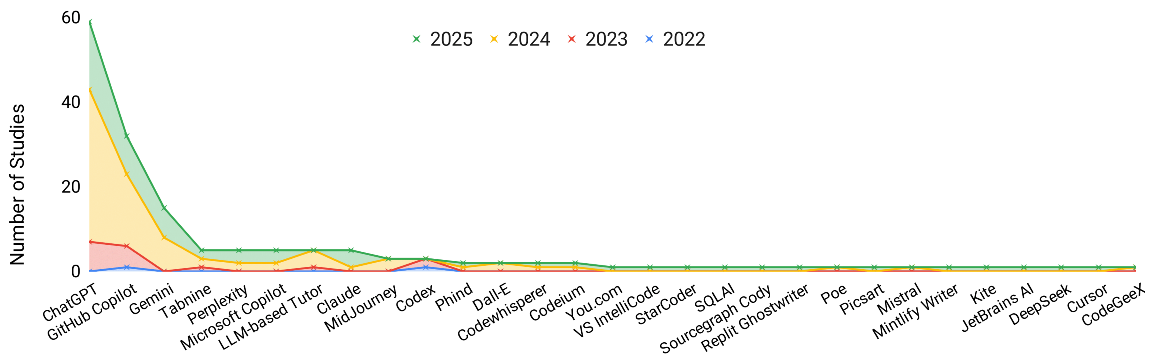


Fig. 5. Distribution of LLM tools over years.

comparing novice developers using open-source and proprietary LLMs. Ahmed et al. [5] found that open-source LLMs can have different performance compared to closed-source LLMs depending on the programming language used. In this context, the study by Pereira et al. [114] appears as an initial comparison between open-source and proprietary in CS/SE Education, where the researchers developed a set of prompt examples to be used by CS students' prompts on ChatGPT and Mixtral[7], and LLaMA2[8], including many software development tasks. However, the authors argue about the need for further exploration with students to identify the benefits of open-source LLMs in SE education. As educational environments often have limited hardware resources, future research could also explore the potential of small and tinny language models - models with a parameter count inferior to 0.8 billion according to Zheng et al. [159] - such as `Qwen2.5-0.5B-Instruct`[9] and `SmolLM2-360M`[10].

5.1 Requirement Engineering and Software Design.

This subsection focuses on how novice software developers utilise LLM tools for tasks associated with Requirements Engineering and Software Design, both of which require considerable creativity from developers [68, 104]. It encompasses tasks such as brainstorming and problem understanding.

5.1.1 Brainstorming. As an old practice with different variations (e.g., visual brainstorming, reverse brainstorming), brainstorming is widely employed in software engineering [27, 130]. Brainstorming is one of the practices to achieve software innovations [112]. Many studies mention novice developers using LLM tools to brainstorm solutions [P: 14, 23, 25, 33, 34, 40, 42, 43, 45, 48, 49, 51, 57, 59, 61, 62, 69, 71, 73, 77]. For instance, Boucher et al. [P: 14] mention novice developers being encouraged to experiment with LLM tools during a summer internship program in Game Development. However, many discontinue using it for brainstorming after the trial. Given its inherent difficulty, it is not surprising that brainstorming is a recurring task in which novice developers use LLM tools. However, Jackson et al. [68] warn that overreliance on LLMs for creative activities might result in losing the *creative intuition*. The findings of the experiment conducted by Kosmyna et al. [83] indicate that frequent AI tool users may experience skill atrophy in brainstorming. In education settings, educators should be aware of this collateral effect in the case that novice developers use LLM tools to brainstorm. In industry settings, developers could be affected by productivity expectations and oversight of these

[7]https://mistral.ai/news/mixtral-of-experts
[8]https://www.llama.com/llama2
[9]https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
[10]https://huggingface.co/HuggingFaceTB/SmolLM2-360M

collateral effects, hindering their skills in the long term [73]. It is the responsibility of software team leaders to see more than productivity metrics and ensure the evolution of their team.

5.1.2 Problem Understanding. Accurate comprehension of the software requirement is a key cornerstone to achieving success in developing high-quality software solutions that satisfy stakeholders' needs [20, 145]. Problem understanding consists of the first step during the problem-solving process, in which programmers interpret and clarify their understanding about the problem [95]. Many studies mention novice developers using LLMs to assist with problem understanding [P: 18, 19, 28, 38, 39, 44, 48, 58, 61, 63, 64, 67, 69, 71]. We observe that all studies mentioning problem understanding involve CS/SE students as study participants. Those situations involve well-defined coding challenges, such as the "More Positive or Negative" coding challenge in the study by Prather et al. [P: 18], which is straightforward to pass all context to the LLM. In real-world scenarios, software developers need to build up the full picture by, for example, breaking the problem into smaller problems, conducting interviews, and asking for further clarification based on their understanding of the domain. With this in mind, LLMs present a potential for industry early-career software to quickly improve their domain expertise (e.g., Finance, Retail), and to develop efficient solutions. However, this potential remains unclear and requires verification.

5.1.3 Others. The variety of possibilities in designing software architecture might make it a challenging task even for experienced developers [28]. Surprisingly, we only identified three studies reporting novice developers using LLMs for architecture definition [P: 38] and diagram generation [P: 29, 63] (UML, sequence diagram, flow chart). Even though a considerable percentage of the study participants is composed of CS/SE students working on small projects, we perceive the small number of papers as a research gap.

5.2 Software Development and Software Quality Assurance.

For tasks associated with Software Development and Software Quality Assurance, we found information retrieval, code generation, conceptual understanding, code understanding, and testing. We provide more details in the following paragraphs.

5.2.1 Information Retrieval. Software developers frequently search the web for software resources, such as documentation and code examples, to support software development activities [63, 133], complementing their lack of knowledge [32]. They can spend up to 20% of their time navigating between web pages [24, 63, 111]. 68.7% of the primary studies mention novice developers using LLM tools for information retrieval related tasks (e.g., [P: 17–23, 35]). For instance, Vasiliniuc et al. [P: 35] report novice developers using LLMs for understanding best practices, discovering new libraries, exploring trade-offs between different libraries, and finding links to relevant tutorials for in-depth learning. In this sense, LLMs appear to work by speeding up the information retrieval process, since developers would not need to browse many pages. However, it also disincentivises content creators from producing relevant content, as they may not be acknowledged or receive the monetising reward through ads [47]. We recommend future studies to explore the perspective of tech content creators (e.g., Dev.to[11] and Medium[12]), focusing on achieving a solution that balances the interests of content creators and AI companies.

5.2.2 Code Generation. Coding is perceived by many developers as a challenging task [13]. For this reason, code suggestion (e.g., code completion and code generation) is a relevant research topic in the software engineering community

[11]https://dev.to
[12]https://medium.com

[33]. AI-code suggestions can save developers' effort by providing personalised code snippets, in comparison with Stack Overflow. Surprisingly, only 56.2% of the primary studies mention novice developers using LLM tools for code generation (e.g., [P: 7–9, 11, 12, 14–22]). We believe the percentage of studies in educational settings influences this value, where CS/SE students would be restricted from using LLMs for code generation. At the same time, although using LLMs for code generation seems positive from the perspective of productivity, there is also the potential for impact on developers' code mental model [94]. Developers' code mental model is developed and changed when developers work on the code base [88]. For early-career junior developers, they initially might struggle with a lack of knowledge involving the code base, but in the long term, that situation would change. We suggest a deep exploration involving the consequences of industrial junior developers using LLMs during companies' onboarding.

The study by Nguyen et al. [110] demonstrated that the correctness of AI-based code varies according to the programming language. Educators should teach novice developers the flaws of LLM tools by also making them work with less prominent programming languages (e.g., Golang, Rust, and Kotlin). There is also the risk that LLMs influence the generation of monocultures [68, 144, 152]. Novice software developers should be aware that there is no *silver bullet* regarding a framework or programming language. Thus, there is a research gap regarding the extent to which LLMs influence the generation of monocultures in the population of novice developers, who may be more susceptible to overrelying on LLMs.

5.2.3 Conceptual Understanding. The foundation of programming extends from basic concepts (e.g., conditionals, looping) to advanced concepts (e.g., design patterns). Among the difficulties faced by CS students identified during their SLR, Qian et al. [118] found difficulty in understanding object-oriented programming concepts. We identified 40% of the primary studies mentioned novice developers using LLM tools for conceptual understanding. This highlights the educational potential held by LLMs [143], supporting novice developers in increasing their understanding of CS/SE concepts and practices. In an industrial context, the fast pace can make it unproductive to ask colleagues for help. However, in an educational context, mentorship interactions have a significant impact on novice developers. In summary, the impact of novice developers adopting LLM tools and their interaction with their mentors remains to be verified.

5.2.4 Code Understanding. Working with unfamiliar code is not an unusual scenario faced by developers [39, 137]. However, when seeking information in the documentation, developers might find its content incomplete, outdated, or incorrect [3]. This is why understanding code, or program comprehension, when the scale is the entire programming [120] - is essential. Qian et al. [118] argue that understanding how code works may be difficult for novice developers. Not surprisingly, 35% of the primary studies report novice developers using LLMs for code understanding. For instance, novice developers in the study by Tabarsi et al.[P: 53] rely on LLMs as a first option while trying to understand code, by copying and pasting the code snippet into ChatGPT. As a potential negative effect, the code-reading skill might not be developed. We suggest that researchers investigate the consequences of LLM adoption on code reading skills.

5.2.5 Testing. Software testing is a vital process to ensure quality and reliability of software systems [142], which are essential to the success of every product [12]. Software testing consists of many tasks, such as test plan, test case preparation, and unit testing preparation [142, 147]. We found novice developers employing LLM tools for unit test generation [P: 2, 8, 21, 31, 42, 43], test case generation [P: 19, 43, 59, 65, 69], data test generation [P: 17, 43], and regression testing [P: 43]. Wang et al. [142] highlight the gap in understanding the capabilities of LLMs in solving software testing problems. At the same time, they found a successful example in the literature combining LLMs with traditional software techniques (e.g., mutation testing, differential testing). For this reason, we recommend that researchers investigate the effects of novice developers combining LLMs and traditional methods.

5.2.6 *Others.*

We also identified novice developers using LLMs for correcting programming language syntax [P: 19, 28, 53, 64, 69, 73, 75, 78]. For novice developers who rely on LLMs for this assistance, they are losing the opportunity to familiarise themselves with the programming language and getting stuck in a state of unfamiliarity with syntax. In industry, there are many situations, like job interviews, where developers might not get external help while coding [16, 71]. We suggest that future research investigate the impact of the performance of LLM users in tech job interviews. There is also a gap in understanding how those LLM users would experience the transition to other programming languages.

5.3 Software Maintenance

In this section, we discuss the tasks related to Software Maintenance, in which novice developers are using LLMs. It includes tasks such as debugging, code refactoring, and code review.

5.3.1 *Debugging.*

It includes detecting, locating, and correcting errors in a software [89]. Traditionally, software developers seek support on online forums, such as Stack Overflow[13], as a common debugging approach [30, 93, 97]. But, even with this support, the debugging process is still a challenging task, especially for novice developers [14, 92]. Not surprisingly, we found 61.2% of the studies reporting novice developers utilising LLM tools for debugging. Novice developers should be aware of limitations regarding LLMs, especially for debugging. From their experimental study, Majdoub et al. [96] found DeepSeek scoring only around 65%. We were surprised to find two studies reporting novice developers using LLMs for *rubber duck debugging*, which is an effective approach to identify the cause of a problem by verbalising how the code works [140, 146]. The literature lacks research on emerging and unique uses of LLMs such as this.

5.3.2 *Code Refactoring.*

This process involves adjusting the software structure without changing its behaviour [108]. This practice helps contribute to enhancing code maintainability and reliability through, for example, removing code duplication and adoption of design patterns [131]. We only found 12.5% of the primary studies reporting novice developers using LLMs for code refactoring [P: 8, 18, 43, 54, 56, 64, 68, 69, 73, 79]. We believe there are two potential reasons: i) code refactoring is not widely employed by novice developers; ii) they might prefer to use traditional code refactoring tools (e.g., SonarQube[14]). Further investigation is required to provide an in-depth explanation.

5.3.3 *Code Review.*

This practice is well-known for its potential to improve the quality of software projects [1, 2, 121]. Usually, code review is conducted by a developer who is not responsible for writing the code under review [82]. We found a few studies mention novice developers utilising LLMs for code review [P: 34, 48, 58, 63, 65, 67, 80]. Sadowski et al. [121] argue that many companies adopted a lightweight code review process focusing on accelerating development. In that sense, LLMs can support novice developers by providing a preliminary code review.

5.3.4 *Others.*

We also found novice developers using LLMs for data analysis of user feedback or test results [P: 31]. They also use LLMs to perform code analysis [P: 5, 19, 54, 65], focusing on performance improvements. While LLMs

[13]https://stackoverflow.com
[14]http://www.sonarqube.org

provide potential for code analysis, novice developers should also employ traditional static and runtime code analysis tools, such as FindBugs [15] and ESLint[16].

RQ2. What key software development tasks are novice developers using LLM-based tools for?

We found evidence of novice developers using LLMs across all SE activities in the software development life cycle, except in Software Project Management. Most of the software development tasks are in Software Development and Software Quality Assurance (e.g., code generation, conceptual understanding). ChatGPT and GitHub Copilot are the most recurrent LLM tools in the studies.

6 RQ3: What are the perceptions of novice software developers on using LLM-based tools?

This section contains the findings regarding the third research question. We first present an overview of novice developers' perceptions in the primary studies. In the following subsections, we present the advantages, challenges, and recommendations reported by novice developers.

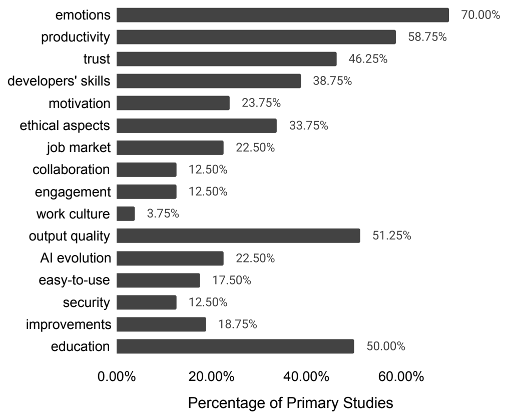
During our analysis, we classified the perceptions of novice developers into four categories: *individual aspects*, *industry collaborative aspects*, *LLM-related aspects*, and *educational aspects*. Individual aspects consist of how LLM influences developers individually: emotions, productivity, trust, developers' skills, and motivation. Industry collaboration aspects refer to software developers working in teams, in the IT industry, using LLMs: ethical aspects such as privacy, copyright, and fairness/bias, job market, collaboration, engagement, and work culture. LLM aspects refer to non-functional and functional LLM features: output quality, AI evolution, user feedback regarding improvements, easy to use, and security. Education-related aspects refer to perceptions involving embracing LLMs in CS Education. We acknowledge that emotions and motivations can potentially influence the team both positively and negatively. Figure 6a shows the distribution of the study participants' perceptions over 77 primary studies. Most studies report participants' emotions (e.g., fear, satisfaction, surprise, pessimism, frustration) towards LLMs and self-reported productivity. We believe that since LLM4SE emerged recently as a research topic, initial research has focused on understanding novice developers' general perception towards LLMs. Researchers could focus on investigating underexplored topics, such as the impact on work culture.

Figure 6b shows the distribution of the study participants' perceived impact of LLM adoption in terms of positive, negative, or mixed. Most studies describe a mix of feelings towards LLMs. However, we also observed a potential negative trend towards LLMs arising from 2024. Figure 7 shows the distribution of study participants' perceptions across the most recurrent LLM tools. Perceptions involving the impact of LLM adoption on self-report productivity, such as automating repetitive and tedious tasks [P: 3], and emotions (e.g., satisfaction and fear) appear commonly together. We observed a research gap in investigations of potential co-variables, such as work culture and satisfaction, influencing LLM adoption.

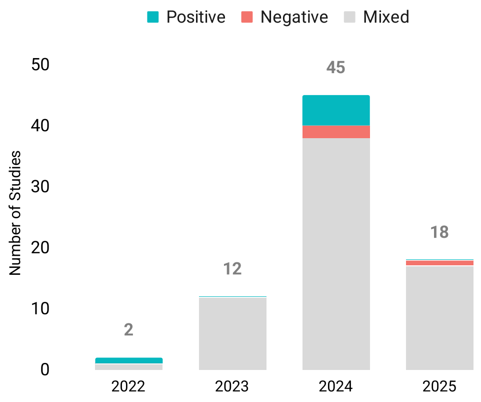
Emotions. Fifty-six studies report emotions related to novice developers adopting LLMs [P: 1–3, 7–10, 12–19, 21–24, 26–30, 32–34, 36, 39–49, 51–55, 57–59, 61, 65, 68, 69, 72, 75–78, 80]. Overall, novice developers show a mix of positive and negative emotions. For instance, one of the participants in the study by Mbizo et al [P: 1] expressed sadness regarding the new generation becoming comfortable using LLMs, and losing valuable learning opportunities previously

[15]http://findbugs.sourceforge.net
[16]https://eslint.org

available while navigating topics on Stack Overflow. Prather et al. [P: 7] observed novice developers genuinely surprised and happy at GitHub Copilot's capabilities. However, they also observed novice developers feeling frustrated or annoyed when GitHub Copilot suggested incorrect or unhelpful long blocks of code. In another study, one of the students from Prather et al. [P: 18] feelt frustrated that GitHub Copilot *"can take away"* from their own thinking. We suggest investigations on the appropriate level of control for AI tools.



(a) Distribution of the topics across primary studies.



(b) Distribution of perceived impact of LLMs over the years.

Fig. 6. Overview of novice developers' perceptions of LLM adoption across 77 primary studies.

Productivity. Forty-seven studies mention perceived impact of LLMs on developers' productivity [P: 1, 2, 4, 5, 7, 8, 10, 12–16, 18, 19, 21, 22, 24, 25, 28, 31, 32, 38–46, 48, 52–54, 59, 61, 63–66, 69, 71, 73–77]. Overall, novice developers perceive LLMs boosting their productivity. We provide more details in section 6.1.1.

Trust. Thirty-seven studies mentioned trust on using LLMs [P: 2, 3, 5, 9–11, 15, 17, 20–22, 25, 27–29, 33, 35, 37–40, 43, 46–52, 55, 59, 60, 62, 64, 68, 69, 78]. Overall, novice developers do not have blind trust in LLMs. For instance, Amoozadeh et al. [P: 3] found that CS students from their survey at two large universities held a neutral view regarding GenAI tools. Additionally, some novice developers in the study conducted by Rogers et al. [P: 11] assume a more extreme position, opting not to use ChatGPT due to their distrust of it.

Developers' skills. Thirty-one studies report the perceptions on the impact of developers' skills [1, 4, 7, 9, 10, 15, 17–19, 21, 23, 25, 26, 29, 32, 34, 37–39, 44, 48, 51, 61, 64, 65, 69–72, 77, 79]. These studies present both positive and negative sides, supporting improvement in developers' skills (See section 6.1.3) and holding developers' skills back (See section 6.2.3). For instance, one of the participants of the study conducted by Mbizo et al. [P: 1] argues that developers who rely on LLMs may never progress since they are skipping the essential learning when using LLMs.

Motivation. Nineteen studies report novice developers' motivations to adopt or not adopt LLMs [P: 3, 8, 9, 14, 24, 26, 32, 34, 39, 43–45, 48, 51, 67, 69, 71–73]. Curiosity, innovative solution, and potential gains in performance are reasons identified by Jaworski et al. [P: 24] for using LLMs. Conversely, Haindl et al. [P: 34] summarise novice developers' reasons for not using LLMs: fear of not developing programming proficiency, incorrect or misleading code, desire to establish their own programming style, fundamental rejection, and scepticism about LLM benefits. Additionally, some participants in the study by Cipriano et al. [P: 66] expressed that they did not use LLMs simply because they did not *"feel the need to"*.

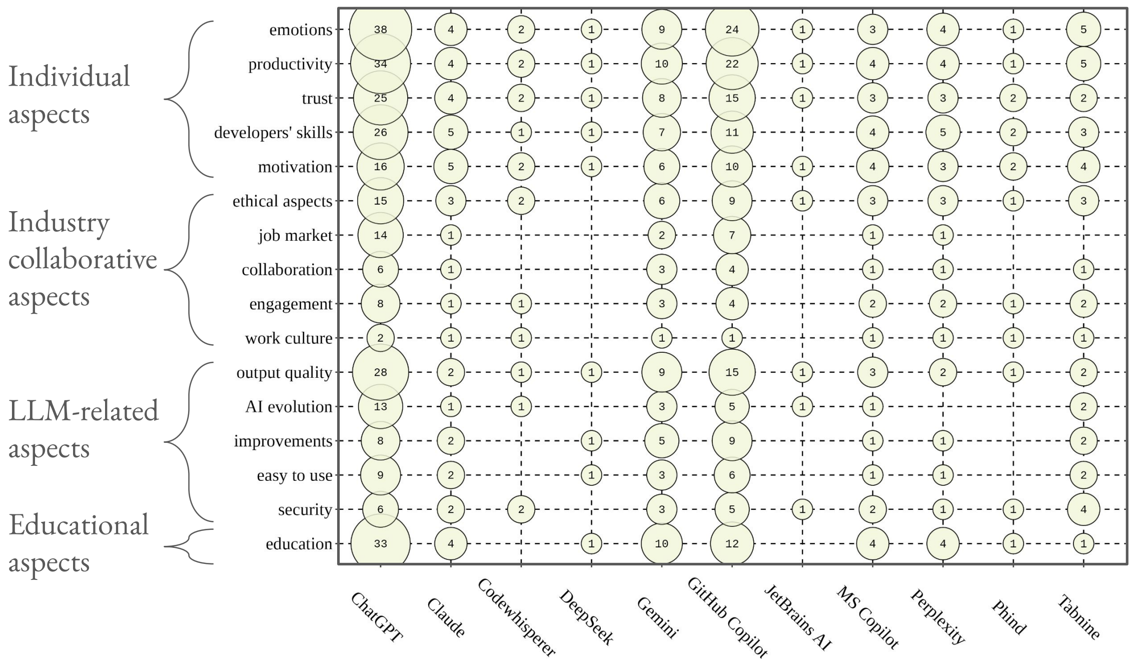


Fig. 7. Distribution of topics discussed by novice developers across most recurrent LLM tools.

Ethical aspects. Twenty-six studies present novice developers' perceptions regarding ethical aspects related to LLMs, such as copyright, fairness, bias, and privacy [P: 1–3, 7, 8, 14, 17, 19, 21, 24, 25, 29, 32, 38–40, 43, 45, 48, 49, 55, 61, 62, 69, 71, 76]. For instance, novice developers in the study by Alpizar et al. [P: 61] expressed concerns about privacy related to inadvertently sharing sensitive or personal data, as well as uncertainty regarding data handling and access by third parties. Participants in the study by Scholl et al. [P: 19] showed concern about bias in AI suggestions.

Job market. Eighteen studies mention perceptions of the LLM impact on the job market [P: 1, 3, 5, 7, 8, 11, 14, 17, 21, 23, 24, 26, 29, 37, 40, 48, 65, 66]. Overall, novice developers express mixed feelings about LLMs, viewing them as both beneficial and potentially harmful to the job market. For instance, Rasnayaka et al. [P: 65] identified some positive perceptions of LLMs as another tool part of the toolset of software engineers. However, Rasnayaka et al. also found some participants expressed strong negative sentiments regarding the future of the software engineering industry, citing concerns about a decrease in programming job opportunities.

Collaboration. Ten studies report perceptions of the implications of adopting LLM on team collaboration [P: 2, 4, 21, 32, 33, 41, 58, 72, 78, 79]. Overall, these studies indicate limitations in the human-AI collaboration. For instance, participants in the study by Sinmaremare et al. [P: 79] pointed out the lack of dynamism during discussions when experimenting AI pair programming.

Engagement. Ten studies report perceptions regarding developers engagement with LLMs [P: 22, 31, 37, 39, 44, 67, 70, 72, 79, 80]. Overall, these studies reveal a mix of perceptions. For instance, Padiyath et al. [P: 70] found that the engagement levels of novice developers are influenced by their career goals and their peers' usage of LLMs. Those who are concerned that LLMs negatively impact their career prefer to avoid LLMs.

Work culture. Three studies report novice developers' perceptions on the impact of LLMs on work culture [P: 1, 39, 52]. One of the participants in the study by Mbizo et al. [P: 1] highlights the bad stigma associated with using LLMs at the workplace. On the other hand, the organisational culture in the company of one of the participants in the study by Li et al. [P: 39] makes them comfortable in using LLMs.

Output quality. Forty-one studies report perceptions regarding quality of LLM output [P: 3–7, 9, 12, 15, 17, 19, 21, 26, 27, 29, 31, 32, 34, 40, 43, 47, 49–51, 54, 55, 58–61, 63, 64, 66, 68–71, 73, 76–79]. One of the participants in the study by Choudhuri et al. [P: 62] highlighted the potential of LLMs provide misleading information by often generating *"plausible yet entirely fabricated information"*. This influences novice developers to develop a sense of distrust towards LLMs. Furthermore, according to novice developers in the study by Korpimies et al. [P: 73], the quality of AI suggestions tends to diminish in more complex situations.

AI evolution. Eighteen studies report perceptions regarding the AI evolution [P: 2, 3, 6, 7, 9, 11, 15, 17, 19, 21, 23, 26, 29, 32, 34, 43, 46, 60]. Although surprised by the capabilities, novice developers also highlight LLM limitations. For instance, participants in the study by Choudhuri et al [P: 46] report ChatGPT (GPT-4) struggles to understand certain problems, provides limited advice on specialised topics, and hallucination.

Improvements. Fifteen studies report novice developers' perceptions of improvements for LLMs [P: 9, 19, 21, 22, 26, 27, 30, 32, 36, 40, 42, 58, 62, 69]. For instance, participants in the study by Yabaku et al. [P: 40] suggest improvements on security features, more customisation options, and better integration with other tools.

Easy to use. Fourteen studies present perceptions related to user experience of LLM tools [P: 4, 8, 14, 19, 21, 23, 26, 32, 40, 42, 45, 52, 69, 77]. For instance, participants in the study by Scholl et al. [19] identified *"intuitive handling, adapting to natural language, multilingual inputs, and even vague inputs"* as positive aspects of their interaction with LLMs. However, they also pointed out that using LLMs may require precise prompting. Yabaku et al. [P: 40] found in their study that ease of use is pointed by novice developers as one of the most appreciated aspects of LLMs for software engineering.

Security. Ten studies have reported on perceptions related to LLMs and security [P: 1, 20, 32, 39, 40, 42, 43, 52, 53, 55]. Overall, novice developers expressed concerns about the potential negative security implications of using LLMs. For this reason, some participants from the study by Tabarsi et al. [P: 53] explain that they *"never had to use it for any security aspects"*.

Education. Forty studies report novice developers' perceptions of LLMs in CS/SE Education [P: 4–7, 10, 17, 19, 23–27, 29, 31–33, 36–38, 41, 47–49, 51, 53, 57, 58, 61, 62, 64–66, 69–71, 73, 74, 76, 78, 79]. Overall, perceptions of adopting LLMs for CS/SE education are mixed, with both positive and negative views (see section 6.1.3 and section 6.2.3). For instance, novice developers in the study by Alpizar et al. [P: 61] agree that LLMs can accelerate the learning process; however, frequent use of these models may also have detrimental effects.

6.1 RQ3a. What are the perceived and experienced advantages of novice software developers on using LLM-based tools?

We found novice developers reporting the benefits of LLM adoption in 65 primary studies. They are organized into benefits into individual aspects, industry collaborative aspects, and educational aspects. We grouped them into four main categories: *gains in productivity and efficiency*, *learning opportunities*, *additional assistance*, and *improvement in code quality*. In most of the primary studies, study participants mention gains in productivity and efficiency. Despite this, LLMs show great potential for novice developers from the perspective of learning opportunities and additional assistance.

6.1.1 Benefits related to Individual aspects. **Gains in Productivity & Efficiency.** We found 45 studies in which study participants self-report the benefits related to improvements in productivity and efficiency. For example, LLMs can generate files in seconds [P: 1, 5, 38], making novice developers complete tasks faster while spending less mental effort and time searching for information [P: 2, 4, 11, 17, 19, 21, 23, 25, 30, 44, 61, 66, 71], typing [P: 14, 75], and troubleshooting [P: 31, 39, 44]. Novice developers can stay focused and avoid distractions [P: 21, 21, 25, 64, 72]. LLMs were seen to automate many repetitive and tedious tasks [P: 3, 7–9, 17, 54, 63, 65], speed up problem solving [P: 18], and reduce the effort to get started [P: 15, 22, 28, 77]. For instance, novice developers can let Copilot generate boilerplate code composed of constructors and simple methods, saving their time [P: 5, 17, 24, 55]. They also mentioned that they can generate code that may require minor changes [P: 30, 32], with minor or no latency on feedback [P: 33, 54, 69, 74, 76]. Thus, novice developers can outsource tasks to LLMs [P: 7]. In this sense, we observe a research gap to extend what tasks novice developers would be comfortable outsourcing to LLMs. We suggest that researchers replicate the study by Masood et al. [98] in the context of LLMs.

Improvement in Code Quality. We found nine studies highlighting potential gains in code quality. LLMs can provide hints of potential improvements on code [P: 5, 40], such as refactoring to become SOLID [P: 8] and more organised [P: 29]. They can also be used to identify minor errors in the code [P: 7, 19, 77]. In summary, we found a small number of references to LLMs improving novice developers' code quality. We believe that this occurs because software developers usually focus on making the code functional, while putting code quality aside. Techapalokul et al. [138] found that CS students maintain an attitude toward software quality even after having gained experience. They recommend that educators teach software quality concepts alongside the fundamentals of computing to convince students regarding the importance of software quality. We see the potential of educators using LLMs to generate sophisticated examples involving *good code* and *bad code* when teaching programming. Future research could explore how the adoption of LLM in education can improve CS/SE students' code quality.

6.1.2 Benefits related to Industry Collaborative aspects. **Additional Assistance**. We identified 15 studies mentioning the potential of LLMs to provide assistance to novice developers. For instance, novice developers can ask *dumb* questions without worrying about other's judgment [P: 19, 71] or bothering colleagues [P: 1, 4, 10, 11, 32, 33, 57]. The constant availability of LLMs also stands out to novice developers [P: 11, 71]. LLMs also support novice developers to understand the reason behind errors [P: 5] and during novel activities [P: 9], guiding them to a solution [P: 34]. LLMs can also support pair programming between novice developers [P: 53, 77]. While analysing the literature, we observed a research gap exploring how LLMs influence novice developers to overcome *impostor phenomenon*. This occurs when individuals have an intense fear that others will perceive them as less capable than they actually are [34]. Novice developers may suffer impostor phenomenon during transition from academia to industry, as highlighted by one of the participants in the study by Zhao et al. [157]: *"I really want to be good enough for my current job... I don't know if I can actually win or become an engineer after all... But now I have mainly high anxiety since I started [this] job... while it appears those who started with me are managing it well and getting stronger"*. Although the current literature has shown emerging studies focusing on software developers (e.g., [56, 135]), Chen et al. [31] point out the limitation in understanding it from a novice developer's perspective, especially in the context of LLMs.

6.1.3 Benefits related to Educational aspects. **Learning Opportunities.** We identified educational-related benefits in 25 studies. LLMs can assist novice developers to learn new coding approaches [P: 1, 6, 18, 19, 23, 25, 64], concepts [P: 3, 29, 47, 64], and programming language syntax [P: 10, 21, 62, 69]. LLMs can also be used as a tutor [P: 19, 22, 27] and can speed up the learning process and reducing boredom [P: 41] via personalised learning [P: 51, 62]. Novice developers can use LLMs to generate examples [P: 59, 61]. They can also learn from the comments in AI-generated code [P: 76]. LLMs were mentioned as having a latent potential for educational applications [P: 49, 52], supporting self-learning in novice developers [P: 53, 76]. However, the extension of the potential of instigating self-learning in early-career junior software developers, as well as supporting them to adjust to industry standards and practices, remains to be verified.

6.2 RQ3b. What are the perceived and experienced challenges & limitations faced by novice software developers while using LLM-based tools?

We identified novice developers reporting challenges and limitations associated with LLM adoption in 68 primary studies. We organised the challenges into individual aspects, LLM aspects, and educational aspects. We grouped them into three main categories: *novice not ready for LLMs*, *losing learning opportunities*, and (LLMs are) *not a good fit for novice developers*. Most studies mention aspects that highlight how novice developers are not adequately prepared to handle the risks of using LLM tools.

6.2.1 Challenges related to Individual aspects. **Novices are Not Ready for LLMs.** We found 35 primary studies reporting aspects showing that novice developers are not mature enough to use LLMs. AI suggestions are based on the trained dataset that might not reflect organisation realities or other niche topics, generating sub-optimal solutions [P: 1–3, 8, 46, 69]. Thus, those tools might generate low quality or incorrect solutions not following, for example, software engineering best practices [P: 6–8, 10, 15, 18, 19, 21, 23, 25, 27, 29–33, 39–41, 43, 45–47, 77]. Novice developers might find it difficult to evaluate the LLM suggestions, which apparently look right [P: 62], due to a lack of background knowledge [P: 10, 25, 26]. They might not know when to use LLMs [P: 60], knowing that LLMs struggle with complex context [P: 47, 77] and might misunderstand users' prompts [P: 71]. To avoid this, novice developers would need to provide very specific and clear prompts to generate a desirable solution, but it is challenging to frame that specific prompt [P: 8, 10, 62, 63]. And when faced with the scenario where the prompt lacks sufficient context [P: 19], a novice might increase context [P: 78]. However, while increasing the context, the response time also increases [P: 77]. They would need to restart the conversation because the model loses context [P: 53] or get stuck in the same incorrect response [P: 19, 78]. They also might be arm-wrestling with LLMs [P: 19, 67], struggling because they do not have enough reasons to refute AI bad code suggestions.

6.2.2 Challenges related to LLM aspects. **Not a Good Fit for Novice Developers.** We identified 12 primary studies reporting aspects showing that LLMs are not recommended for novice developers. Developers would need to customise LLMs to avoid, for example, ChatGPT generating code with unnecessary comments [P: 76], or force it to match codebase style [P: 22, 62]. However, novice developers might not know how to adjust complex parameters (e.g., temperature) [P: 49]. There is a concern that novice developers would upload proprietary code into LLMs [P: 1, 24, 42]. AI can be misleading, for example, through AI sycophancy, which is another trap novice developers could fall into [P: 6]. More than demonstrating friendliness, *AI sycophancy* [127, 136] makes LLMs align their responses with users' perspectives even though users provide incorrect views. In this sense, Bo et al. [23] propose to adjust LLMs to act more actively to correct users' faulty assumptions. The practical applicability of such approaches remains to be verified.

6.2.3 Challenges related to Educational aspects. **Losing Learning Opportunities.** We found educational-related challenges in 31 primary studies. By outsourcing tasks to LLMs, novice developers would miss essential learning experiences [P: 1, 9–11, 19, 25, 26, 29, 32, 35–37, 44, 49, 54, 55, 57, 61, 65, 71, 79]. For instance, ChatGPT might provide a full solution rather than a partial solution during learning [P: 57]. Overreliance on LLMs might make a novice developer dependent [P: 2, 23] and lazy [P: 23, 72, 79], rather than eager to learn. This attitude could hinder their potential of mastering programming languages and concepts [P: 70]. By just copying and pasting AI suggestions, they also lose the opportunity to understand the codebase [P: 7], which is a foundation to posterior debugging processes [P: 22]. Novice developers might spend a lot of time creating very descriptive prompts but not understand the reason behind the AI response, similar to a *blackbox* [P: 4, 25]. There are also the potential negative implications on developers' confidence in their intuition [P: 2], where the developer would trust LLM suggestions over their own, disturbing their thinking process [P: 7, 9, 15, 62]. We suggest future research in-depth explorations on the impact on developers' confidence, which might influence developers' productivity. The inconsistency of LLM suggestions [P: 19, 54], by providing different responses to the same prompt, also provides a challenge for learning.

Others. Novice developers also mentioned that code generation takes away the best part of software development: coding [P: 65]. LLM adoption may also lead to a loss of sense of ownership [P: 76]. Bird et al. [19] found that code ownership has a strong relationship with software quality. Thus, while adopting LLMs, there is a potential to increase the number of system failures. There is also a loss of mentorship, where incoming novice developers might not seek support from their seniors [P: 72], as well as the increasing expectations of employers [P: 17]. We suggest future research survey software team leaders and IT managers to get their perceptions. In addition, when seeking to use high-performance models, they may require payment [P: 71].

6.3 RQ3c. What are the recommendations & best practices suggested by novice software developers while using LLM-based tools?

We found novice developers reporting the best practices regarding LLM adoption in 34 primary studies. We organised the recommendations into LLM aspects and industry collaborative aspects. We grouped them into three main categories: *prompt engineering*, *cautious towards LLMs*, and *when to use LLMs*. Most of the selected studies include recommendations to developers to be analytical towards AI suggestions. This shows that novice developers have a prudent attitude regarding the LLM adoption. Most of the recommendations compiled below are perhaps applicable to software developers at all levels of expertise, except those regarding when to use LLMs, which are more specific to novice developers.

6.3.1 Recommendations related to LLM aspects. **Prompt Engineering.** We identified study participants using a prompting engineering approach in ten studies. For instance, break prompts into small tasks [P: 9, 10, 53, 69], provide context in prompt [P: 68], asking for specific code segments [P: 14]. Study participants also suggest asking follow-up questions to guide LLM towards the solution [P: 26, 32, 36]. Study participants also highlight the importance of learning prompting engineering practices [P: 54, 65, 69], and prompting in English as a strategy to improve the accuracy of LLMs [P: 54]. However, novice developers with English as a second language might face a language barrier [P: 79]. These results indicate that there is an emerging understanding by novice developers about the importance of prompting engineering when using LLMs.

6.3.2 Recommendations related to Industry Collaborative aspects. **Cautious towards LLMs.** We found that nineteen primary studies mentioned a cautious attitude towards LLMs. Potential negative consequences for misuse of LLMs include, for example, disruption of the developer's mental coding flow [P: 21]. In this sense, study participants suggest that novice developers should adopt an analytical attitude towards LLMs, evaluating each AI suggestion [P: 2, 18, 33, 52, 64] and modifying it to their context. They recommend the adoption of tools from reliable AI vendors with a focus on security, and the use of external platforms (e.g., documentation, Stack Overflow, code scanning) to double-check AI suggestions before accepting them [P: 8, 9, 32, 53, 55, 61, 62, 68, 69]. They also recommend that novice developers treat

AI-generated code as a baseline [P: 31, 64], not requesting to generate the entire solution, because it may generate wrong code [P: 3, 18, 29]. Study participants suggest novice developers collaborate with other developers to improve their understanding of those tools [P: 6]. There is also a recommendation to not use ChatGPT for system-wide testing due to security concerns [P: 53], since LLMs struggle with complex tests.

When to use LLMs. We found nine studies mentioning recommendations to help novice developers decide when to use LLM tools. Study participants recommend novice developers only go after LLM tools when they fail to reach a solution by themselves [P: 17, 36, 60, 61, 64, 74]. LLMs can also be used to improve novice developers' code quality [P: 53]. Focus on improvements, participants of the study by Alkhayat et al. [P: 6] suggest that Virtual Reality developers familiarise themselves first with the Unity interface and the C# programming language to be able to comprehend ChatGPT suggestions. In that sense, study participants also recommend novice developers use LLMs as a learning tool to support them in solidifying the basis [P: 69, 70].

RQ3. What are the perceptions of novice software developers on using LLM-based tools?

A majority of the studies present a mix of positive and negative aspects related to novice developers adopting LLM tools. Gains in productivity appear as the most commonly perceived advantage, while many study participants in the primary studies recognise aspects where LLMs and novice developers are not a good fit. Regarding recommendations, most of the studies suggest a cautious approach towards LLMs.

7 RQ4. What are the limitations and recommendations for future research that we can distil based on the primary studies?

This section contains the findings regarding the fourth research question. We first present an overview of study limitations and future research needs suggested in the primary studies.

7.1 Primary Study Limitations

We categorised the primary study limitations into three main categories: *limitations in approach*, *limitations in data collection and analysis*, and *limitations in findings*. Most studies report limitations in data collection and analysis that are not directly related to LLMs.

Limitations in Approach. We identified limitations regarding the study approach in thirteen primary studies. This includes limitations in the design of the task [P: 2, 4, 12, 13, 30], where confounding variables [P: 34, 36, 37, 42, 46], such as the level of familiarity with AI tools, could impact the results. A few studies recognise the probabilistic nature of LLMs during experiments, difficulty in study replication [P: 15, 66]. Other studies acknowledge that the non-realistic experimental environment could impact the findings [P: 7, 15, 20, 30, 36, 76, 78, 80]. Another study acknowledged a potential deterioration in the performance of used LLMs because the study participants prompted the LLMs in German, their native language, instead of English [P: 60]. This understanding is consistent with the literature that demonstrates that ChatGPT shows poor performance for prompts in languages other than English [87, 156].

Data Collection & Data Analysis. We identified limitations in data collection and analysis in 46 primary studies. For example, limitations in sampling (e.g., participants' geographic location, number of participants), which could not reflect the opinion of the entire SE population, are a common limitation to most empirical SE studies [P: 1–3, 6, 8–10, 13, 14, 16, 18, 20, 25, 29, 30, 32–34, 37–39, 41–46, 52, 53, 55, 56, 58, 62, 66, 68, 71–79]. They also acknowledge potential self-selection bias where, for example, participants interested in the study topic would be more interested in joining their studies [P: 10, 37, 46, 55, 56, 60, 62]. Participants could also misinterpret researchers' questions [P: 32, 34, 42]. Regarding data analysis, researchers acknowledge potential research bias during the qualitative data analysis process [P: 59, 60, 73]. All these reported study limitations are common in empirical studies.

Limitations in Findings. We identified 25 primary studies reporting limitations regarding findings. This includes self-report data [P: 1, 3, 19, 32, 36, 37, 49, 52, 54, 55, 58, 61, 69, 71–73] and memory bias [P: 3, 32, 42, 62]. A few studies recognise that findings related to ChatGPT and Copilot could not be generalised to all LLM tools [P: 2, 18, 22, 66]. A few studies recognise that their findings do not converge over a long period of observation and experiment [P: 2, 9, 29, 37, 77]. Only a few studies recognise that the fast pace of evolution of AI tools is making study findings outdate more quickly [P: 9]. In that sense, we suggest that future research indicate LLM versions or features available when the study was conducted to enable readers to understand LLM capabilities during that period.

7.2 Future Research Needs

We found 76 primary studies proposing future investigations. We grouped them into five main categories: *exploratory studies*, *development and improvement of guidelines and tools*, *replication studies*, *extension studies*, and *longitudinal studies*. Most of the primary studies suggest topics for future exploratory studies.

Exploratory studies. We identified key recommendations for research explorations in 30 primary studies related to novice developers. For instance, Boucher et al. suggest that researchers explore to what extent LLM tools can support game development [P: 14]. They also recommend more studies exploring attitudes (e.g., resistance) in early career software professionals [P: 14, 29, 73], taking in consideration gender minority [P: 17], and communities and organisational settings [P: 17]. Investigate better approaches for developers to understand long blocks of generated code [P: 9, 21], as well as ways to better control LLM tools [P: 9, 23]. Future studies could explore how LLM tools influence pair programming [P: 79] and novice developers' code reuse [P: 60]. There are recommendations for research focusing on organisational aspects (e.g., culture, size) and how they affect the adoption by software practitioners. [P: 39, 70]. Studies also suggest research on how group dynamics in software development teams with developers with multiple levels of expertise are affected by LLM adoption [P: 6, 15, 33, 39]. In this context, one of the participants in the study by Kemel et al. [76], which conducted an exploratory case study of 7 European companies, mentioned that: *"For me, when I started, I had some difficulty in making some code contributions because the code base was so huge. And it took... a lot, a lot of time to get used to it. So if I, if I had [GitHub] Copilot back then, I think I would have made contributions much earlier"*. Based on this, we suggest future studies seek to understand how junior developers can leverage LLM tools to improve their understanding of large code bases.

In education, many studies suggest more investigations on how LLM tools can be incorporated into computer education, aligning with educational objectives and learning outcomes [P: 5, 7, 11, 12, 23, 29, 36, 39, 44, 46, 47, 64, 67], understanding which core skills expected of graduates [P: 13]. But also, focused on adjusting assessment to properly evaluate students in an LLM era, mitigating the risk of them using LLMs to cheat [P: 4, 6, 11]. They also suggest exploration the impact on novice developers' learning and motivation while being exposed to numerous LLM suggestions [P: 3, 4, 18, 26], how novice developers break down tasks and write prompts for LLMs [P: 4], and comparison between traditional learning and AI-enhanced learning with student traditional learning, leveraged with personalised student assistance [P: 46, 54, 60]. In this context, there is also a suggestion for exploring the impact of LLM tools on individual versus team-based learning [P: 54].

Development and Improvement of Guidelines and Tools. We identified ten primary studies discussing ideas for LLM tools and guidelines. For instance, studies suggest designing new metrics to improve the alignment of LLM tools with individual preferences [P: 20, 23, 80]. Regarding Computer Education, they suggest the development of specialised GPT models focused on educational aspects that could be incorporated into well-known educational environments [P: 19, 38, 56], supporting feedback across submissions [P: 27]. In this sense, ChatGPT provides a collection of customised GPT models[17] that CS/SE educators can use. Educators can also create their own customised GPT model by following the steps in the study by [72], which created a neurosurgical research paper writer and medi research assistant. To support effective adoption of LLM in Computer Education, it is also necessary to develop an easy way to cite AI support in students' code [P: 57]. Given the necessity to understand how LLM tools work, Barke et al. [P: 9] suggest that the developer community work collaboratively to create a "community guide", including best prompts and comments used to achieve the best results. We found the subreddit `r/ChatGPTPromptGenius`[18] that includes many relevant discussions that help to understand the behaviour of ChatGPT. Future studies could explore submissions on this subreddit, similar to when Kuutila et al. [86] explored the subreddit `r/ProgrammerHumor`. To improve understanding of debugging, Akhoroz et al. [P: 69] suggest LLM tools to include visual explanation. One of the studies recommends the development of guidelines for the responsible use of LLMs in the industry [P: 45]. Berengueres et al. [17] explored the ethical aspects related to how LLM service providers could regulate their LLM services, but a research gap remains regarding practical guidelines.

Replication studies. We found nine primary studies suggesting replication of their empirical studies. Replication of studies is a valuable strategy adopted by the research community for over 30 years, aiming to verify and compare the findings of previous studies [18, 38]. Studies recommend validating findings in other educational settings [P: 26, 51, 59, 70] and other areas [P: 19]. They also recommend that future studies update their work with recent LLM versions [P: 13, 33]. For instance, the study by Gardella et al. [P: 13] was conducted using the GitHub Copilot version that did not include a chat mode for code generation, but only a comment-based code generation.

Extension studies. We found twenty-six primary studies recommending future research to extend their studies. For instance, they suggest expanding the studies by incorporating other LLM tools [P: 2, 18, 49, 54, 60], different tasks [P: 13], and different projects [P: 63], as well as more diversity and larger population of participants [P: 1, 4, 10, 13, 18, 19, 24, 30, 33, 34, 37, 41, 42, 45, 53, 64, 68, 73, 76, 77], and different mixed-method approaches [P: 2]. This is consistent with the current landscape, where there is a greater diversity of LLM tools nowadays, such as Grok[19] and Qwen[20], compared to 2022 when ChatGPT became recognised worldwide. They also suggest conducting studies with long-term LLM users [P: 15, 24] and real-world settings [P: 30].

Longitudinal studies. We identified fourteen primary studies recommending longitudinal research on the impact of LLM adoption in general, as well as on novice developers' learning [P: 13, 64, 75]. This challenging approach, which involves data collection related to a long time frame, has been utilised by SE researchers for a long time (e.g., [50, 77]). While facing a massive amount of potential data, it is essential to efficiently select the most appropriate metrics to be collected. In our primary studies, authors suggest the following metrics: perceptions and challenges over time [P: 1, 7, 23], and adoption patterns and trends [P: 8, 52, 69]. It is also necessary to clearly delimit the scope, for example, by investigating the effects of an evidence-based software engineering training [116]. Studies suggest that future long-term investigations are required on the implications of adopting LLM tools in pair programming [P: 79] and career readiness

[17]https://chatgpt.com/gpts
[18]https://www.reddit.com/r/ChatGPTPromptGenius
[19]https://grok.com
[20]https://qwen.ai

[P: 65]. They also recommend long-term studies involving multiple organisations [P: 37] and real-world large software projects [P: 30], similar to Cedrim et al. [29], which conducted a longitudinal study on the impact of refactoring on code smells using 23 software projects. We observe that future research could contribute by building a large dataset that contains the interactions of novice developers with LLMs for exploration by the research community.

RQ4. What are the limitations and recommendations for future research that we can distil based on the primary studies?

We identified limitations across three main categories: limitations in data collection and analysis (57.5%), limitations in findings (31.2%), and limitations in approach (16.2%). We also identified the reported future research needs, classifying them into five main categories: exploratory study (37.5%), extension study (32.5%), longitudinal studies (17.5%), development and improvement of guidelines and LLM tools (12.8%), and replication study (11.25%).

8 Discussion and Research Roadmap

In this section, we discuss the SLR findings in terms of implications for software developers and educators. Then, we synthesise the research directions for future research.

Implications for Practice. We discussed in section 5 the reported SE tasks that novice developers are using LLMs. Most of the SE tasks go under software development and software quality assurance activities. In the context of novice developers, there is a greater potential for using LLMs for concept understanding, which would improve their skills in the long term. At the same time, there are many recommendations for novice developers to act with caution, especially when they are consolidating the fundamental CS concepts. In this sense, the key strategy is to understand when to use LLMs (e.g., after having gained familiarity with the programming language syntax) and how to use LLMs (i.e., prompt engineering). Software team leaders have the responsibility of looking after their team members, alerting them about the potential LLM pitfalls (e.g., introducing bugs or non-functional code) [49]. They are also responsible for creating an environment that embraces team collaboration, vital to the success of software development [134]. According to Strode et al. [134], *Working in personal caves* hinders team effectiveness by potentially affecting the common understanding of the process. By overrelying on LLMs, novice developers could underappreciate team collaboration. Software team leaders should monitor those developers and promote team spirit.

Implications for Education. Given the potential consequences (e.g., disruption of the system in the production environment due to AI-generated code introducing errors), novice developers should be clearly instructed about the limitations of using LLMs. We suggest that educators present the limitations of LLMs, for example, how their performance depends on the context (e.g., programming language, task). For this reason, educators could incorporate realistic scenarios and projects that students would face in industry using role-play and simulations. For instance, a CS/SE student would act as employees from a company in a domain very concerned about privacy and data protection, such as finance and government. In this context, Zheng et al. [158] conducted an evaluation of different LLM tools, using a dataset that includes 12 application domains. They reported that famous LLMs often perform poorly in domain-specific code generation tasks. Another example, the students could work on legacy projects in less prominent programming languages (e.g., Ruby, PHP) or programming languages in which popular LLMs perform poorly.

8.1 Directions for Future Work

In previous sections, we have indicated several research gaps while presenting the findings. We synthesise all these research opportunities in Table 7. In the rest of this section, we present an additional set of recommendations to the SE research community for future exploration.

Novices & Vibe Coding. We are surprised that vibe coding did not appear in the publications in 2025, even though there is a large repercussion in the software industry [58]. The term *vibe coding* refers to an emerging new programming style where developers completely rely on LLM tools to generate good quality code using natural language, while developers disconnect from activities directly related to code (e.g., code writing, code reading) [124]. Based on the potential of vibe coding, there are discussions on Reddit (e.g., [8, 9]) exploring the implications of vibe coding to software developers. From a novice developer's perspective, vibe coding reinforces the idea of AI replacing software developers. However, experienced developers understand that code development is only one of the phases of the software development life cycle. There is also the hidden cost (e.g., power and water consumption) of using LLM tools [129]. In this sense, we recommend investigations regarding the impact of vibe coding on novice developers.

LLMs for Software Project Management. As mentioned in Section 5, there is a research gap in software management. Software management encompasses practices to supervise activities across the software development life cycle [102]. Zhang et al. [156] identified during their systematic survey the potential of project managers to employ LLMs to analyse developers' emotions [66], such as *frustration* due to postponing merging pull requests. Previous studies explored the connection between developers' emotions and their productivity [36, 107, 151]. With this in mind, we recommend that researchers explore the extent to which LLMs can improve project managers' support for early-career developers. In this context, we also observe a research gap in studies focused on the impact of LLM tools on the mental health of novice developers. Our suggestion aligns with a suggestion in the position study by Meem [100] that proposes that researchers investigate what features of LLM tools can be helpful or harmful for the mental health of software practitioners. Another motivation for this research includes the findings from the study by Graziotin et al. [54, 55], which identified from their survey with 181 developers that mental disorders (e.g., anxiety, depression) cause delays in the software development process.

Ethnography Studies with Early Career Developers. During our analysis of the research methods employed in the primary studies (See Fig. 4a), we observe that there is no paper using ethnography as a research method. According to Sharp et al. [128], SE researchers can leverage ethnography to not only investigate what software practitioners do but also their motivation behind it. However, although this research method is widely employed in Computer-Supported Cooperative Work [22] and Human-Computer Interaction [21], it is still not widely adopted by SE researchers, as traditional research methods (e.g., interviews and questionnaires) are [128]. In the context of LLMs, de Seta et al.[40] proposed *synthetic ethnography* as an adaptation of the traditional ethnography focused on qualitative studies of LLMs. We suggest that future research follow this ethnography variation, as exemplified in the study by Holmquist et al. [62].

LLMs Customised to Novice Developers' Needs. We presented in Section 6 many positive and negative aspects associated with novice developers using LLM tools. However, we also observed that current LLM tools do not provide mechanisms to effectively lead novice developers in improving their skills, as they can simply copy and paste AI suggestions. For this reason, we suggest that future studies investigate approaches that combine educational potential with principles of productivity and efficiency, which are essential in industry. Otherwise, if these mechanisms are not developed and novice developers become overly reliant on LLM tools, the improvement of their skills (e.g., critical thinking [90], creativity [85]) could be hindered.

Table 7. Synthesis of research gaps from previous sections 4, 5, and 6.

ID	Research Gap	Description
1	LLMs in Domains with Privacy Restrictions	To explore the use adoption of LLMs in domains that impose restrictions on privacy, such as government, finance, and healthcare. (See section 4.1)
2	Studies in Industry Settings	Most of the studies were conducted in university settings. There is a need for case studies in industry settings. (See 4.2)
3	LLMs supporting Effort Estimation and Task Prioritisation	To explore the potential for LLMs support novice developers in effort estimation and task prioritisation. (See section 5)
4	Open-source LLMs in SE Education	To investigate the potential benefits of novice developers using open-source LLMs in SE education, particularly in environments that are often constrained by hardware resources. (See section 5)
5	LLMs and Domain Expertise	To study the potential influence of LLMs on junior developers quickly improving their domain expertise, such as finance and retail. (See section 5.1.2)
6	LLMs and Architecture Definition	To explore the potential of LLMs to improve novice developers' understanding of architecture definition. (See section 5.1.3)
7	Content creators and AI companies	To investigate the perspective of tech content creators, focusing on achieving a solution that balances the interests of content creators and AI companies. (See section 5.2.1)
8	LLMs and Novices' Companies' Onboarding	To study the impact of junior developers using LLMs during companies' onboarding. (See section 5.2.2)
9	LLMs influencing Monocultures	To explore the potential influence of LLMs on the generation of monocultures in novice developers. (See section 5.2.2)
10	LLMs on Mentorship Interactions	To explore the consequences of LLM adoption on mentorship interactions between senior and novice developers. (See section 5.2.3)
11	LLMs and Developers' Code Reading Skills	To investigate the consequences on novice developers' code reading skills by adopting LLMs. (See section 5.2.4)
12	LLMs and Job Interviews	To study the impact of LLM users in tech job interviews. (See section 5.2.6)
13	LLMs and Programming Familiarisation	To explore how LLM users would experience migrating to other programming languages. (See section 5.2.6)
14	Unique and Emerging LLM Uses Cases	To investigate interesting use cases of using LLMs by novice developers, such as for simulating user interaction. (See section 5.3.1)
15	LLMs for Code Review	To study the use of LLMs by novice developers for code review. (See section 5.3.2)
16	LLMs in Under-explored Topics	To investigate the potential consequences of LLM adoption in underexplored topics, such as the impact on work culture. (See section 6)
17	Co-variables on LLM adoption	To explore the potential co-variables on LLM adoption, such as work culture and satisfaction. (See section 6)
18	Appropriate Level of Control for AI tools	To explore different types of user experience provided by current AI tools, seeking to understand the appropriate level of control. (See section 6)
19	Outsourcing to LLMs	To study what potential software development tasks novice developers would be comfortable outsourcing to LLMs. (See section 6.1.1)
20	LLMs Instigating Self-Learning	To investigate the potential of promoting self-learning in junior software developers (See section 6.1.3)
21	LLM and Impostor Phenomenon	To explore the influence on LLMs to support novice developers overcome impostor phenomenon (See section 6.1.2)
22	LLMs in Software Quality Education	To study how the adoption of LLM in education can improve CS/SE students' code quality. (See section 6.1.1)
23	Experiment with LLMs	To investigate the applicability of adjusting LLMs to act proactively in correcting novice developers' mistakes. (See section 6.2.2)
24	Software Team Leaders' Expectations	To explore software team leaders' and IT managers' expectations for novice developers in an LLMs era. (See section 6.2.3)

9 Threats to Validity

Although this SLR follows the guidelines presented by Kitchenham and Charters [10, 80], and the methodology is the result of many discussions involving experienced researchers, it is essential to acknowledge potential threats to the validity of the results (e.g., missing relevant papers). Ampatzoglou et al. [7] highlight that the empirical SE community adopts Wohlin et al.'s approach [149] (*conclusion*, *internal*, *construct*, and *external validity*) for quantitative research within software engineering.

Internal Validity. It examines whether a study condition makes a difference or not, and whether there is sufficient evidence in the data collected that supports the study conclusions [7, 149]. This study methodology was initially developed by the first author, but it was also reviewed by the other co-authors and SLR experts before starting the study. Then, the first author conducted a pilot test for the paper selection, which enabled the identification of the need for adjusting the search string and inclusion and exclusion criteria after validating with the other authors. The search string was customised for each one of the 7 data sources, being built on several attempts targeting to optimise the results. We acknowledge our search string may not have included relevant terms such as "Hugging Face" and "language model", which could have included the most well-known portal for language models and other types, such as "small" and "tiny" language models. Including these terms could have led to the discovery of more essential portals for language models, including specific types like "small" and "tiny" language models. This would have potentially broadened the search for relevant primary studies on the topic and strengthened adherence to the term "Large Language Models (LLMs)" within the intervention. Our SLR protocol encompasses various rounds of filtering the studies, focusing on minimising the selection bias. A pilot test was also conducted for the data extraction process, which enabled us to adjust after identifying other relevant data that could be extracted from the studies.

Construct Validity. It refers to how well constructed the methodology is, being effective to what was intended [7, 149]. Our SLR protocol combines different strategies to ensure that we are collecting relevant studies: a) primary search: involving the six most relevant digital libraries; b) secondary search (*forward* and *backward snowballing* as secondary search); c) search in the grey literature (i.e., arXiv). The pilot test for paper selection and data extraction also helped to evaluate and improve the SLR protocol (e.g., inclusion and exclusion criteria), making the necessary adjustments. Finally, continuous discussions between the authors (and other SLR experts) helped to build a well-defined SLR protocol while reducing threats.

Conclusion Validity. This aspect refers to which extension the conclusions can be reached from the data collected [7, 149]. Focusing on a solid data extraction process, our data extraction form was developed based on our RQs, which were developed using the PICOC framework. The SLR data, including the spreadsheets with the stages accomplished throughout this SLR, allow other researchers to replicate this study.

External Validity. It examines whether the findings can be generalised [7, 149]. Our search was carefully adjusted to include publications from 2022, aligning with ChatGPT's public release. However, we did not limit our selection to studies using ChatGPT specifically. This SLR also follows Naveed et al. [109] while restricting our search to publications in the English language because it is broadly adopted for reporting research studies. We also excluded vision and opinion papers since they may highlight an individual perspective and lack generalisation. Finally, our findings were based on 80 studies - a reasonable amount considering the empirical software engineering literature (e.g., [60, 78, 109]). Additionally, we acknowledge that papers published between August 2024 and the date the paper was submitted (July 2025) were excluded from the systematic literature review.

10 Conclusion

We conducted a systematic literature review on novice software developers' perspectives on the adoption of LLM tools, which resulted in the selection of 80 relevant studies by following the guidelines provided by Kitchenham et al. [10, 80].

Our analysis of our primary studies indicates that: 1) there is a fast growth in publications in LLM4SE; 2) Although initially, research focused basically on ChatGPT and Copilot, more diversity in LLMs is emerging in the recent publications; 3) there are few papers focused on novice developers in industrial settings; 4) Participants in most of the selected studies have a mix of positive and negative perceptions about the impact of adopting LLM tools.

In RQ1, we identified the motivations and methodological approaches of the studies. Interviews and questionnaires are the research methods more commonly adopted in the primary studies. In RQ2, we identified the software development tasks and LLMs used by novice developers in the studies. While most of the primary studies mention SE tasks related to software development and software quality assurance, there is a research gap in studies that explore software management-related tasks. In RQ3, we identified a variety of topics discussed in the studies (e.g., emotions, output quality) as well as benefits, challenges, and recommendations. Not surprisingly, gains in productivity and efficiency are the most commonly reported benefits. We found many challenges potently indicating that novices are not ready for LLMs, and many recommendations suggesting that developers be cautious when adopting these tools. Finally, we present the study limitations and future research needs in RQ4. Most of the studies have limitations regarding data collection and analysis, and exploratory studies are suggested for future investigations. For future research, we intend to investigate the research gaps presented in section 8 (See Table 7).

Acknowledgements

Ferino is supported by a Faculty of IT Post-graduate Scholarship. Grundy was supported by Australian Research Council Laureate Fellowship FL190100035.

Appendix A. Primary Studies

[P1] Takura Mbizo, Grant Oosterwyk, Pitso Tsibolane, and Popyeni Kautondokwa. Cautious optimism: The influence of generative ai tools in software development projects. In *Annual Conference of South African Institute of Computer Scientists and Information Technologists*, pages 361–373. Springer, 2024.

[P2] Crystal Qian and James Wexler. Take it, leave it, or fix it: Measuring productivity and trust in human-ai collaboration. In *Proceedings of the 29th International Conference on Intelligent User Interfaces*, pages 370–384, 2024.

[P3] Matin Amoozadeh, David Daniels, Daye Nam, Aayush Kumar, Stella Chen, Michael Hilton, Sruti Srinivasa Ragavan, and Mohammad Amin Alipour. Trust in generative ai among students: An exploratory study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 67–73, 2024.

[P4] Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–23, 2023.

[P5] Arne Styve, Outi T Virkki, and Usman Naeem. Developing critical thinking practices interwoven with generative ai usage in an introductory programming course. In *2024 IEEE Global Engineering Education Conference (EDUCON)*, pages 01–08. IEEE, 2024.

[P6] Amany Alkhayat, Brett Ciranni, Rupa Samyukta Tumuluri, and Rohit Srinivas Tulasi. Leveraging large language models for enhanced vr development: Insights and challenges. In *2024 IEEE Gaming, Entertainment, and Media Conference (GEM)*, pages 1–6. IEEE, 2024.

[P7] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. "it's weird that it knows what i want": Usability and interactions with copilot for novice programmers. *ACM Transactions on Computer-Human Interaction*, 31(1):1–31, 2023.

[P8] Daniel Russo. Navigating the complexity of generative ai adoption in software engineering. *ACM Transactions on Software Engineering and Methodology*, 2024.

[P9] Shraddha Barke, Michael B James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA1):85–111, 2023.

[P10] John Chen, Xi Lu, Yuzhou Du, Michael Rejtig, Ruth Bagley, Mike Horn, and Uri Wilensky. Learning agent-based modeling with llm companions: Experiences of novices and experts using chatgpt & netlogo chat. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–18, 2024.

[P11] Michael P Rogers, Hannah Miller Hillberg, and Christopher L Groves. Attitudes towards the use (and misuse) of chatgpt: A preliminary study. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 1147–1153, 2024.

[P12] Wei Wang, Huilong Ning, Gaowei Zhang, Libo Liu, and Yi Wang. Rocks coding, not development: A human-centric, experimental evaluation of llm-supported se tasks. *Proceedings of the ACM on Software Engineering*, 1(FSE):699–721, 2024.

[P13] Nicholas Gardella, Raymond Pettit, and Sara L. Riggs. Performance, workload, emotion, and self-efficacy of novice programmers using ai code generation. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, page 290–296, New York, NY, USA, 2024. Association for Computing Machinery.

[P14] Josiah D Boucher, Gillian Smith, and Yunus Doğan Telliel. Is resistance futile?: Early career game developers, generative ai, and ethical skepticism. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2024.

[P15] Thomas Weber, Maximilian Brandmaier, Albrecht Schmidt, and Sven Mayer. Significant productivity gains through programming with large language models. *Proceedings of the ACM on Human-Computer Interaction*, 8(EICS):1–29, 2024.

[P16] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Measuring github copilot's impact on productivity. *Communications of the ACM*, 67(3):54–63, 2024.

[P17] James Prather, Paul Denny, Juho Leinonen, Brett A. Becker, Ibrahim Albluwi, Michelle Craig, Hieke Keuning, Natalie Kiesler, Tobias Kohn, Andrew Luxton-Reilly, Stephen MacNeil, Andrew Petersen, Raymond Pettit, Brent N. Reeves, and Jaromir Savelka. The robots are here: Navigating the generative ai revolution in computing education. In *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '23, page 108–159, New York, NY, USA, 2023. Association for Computing Machinery.

[P18] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. The widening gap: The benefits and harms of generative ai for novice programmers. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*, pages 469–486, 2024.

[P19] Andreas Scholl and Natalie Kiesler. How Novice Programmers Use and Experience ChatGPT when Solving Programming Exercises in an Introductory Course . In *2024 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Los Alamitos, CA, USA, October 2024. IEEE Computer Society.

[P20] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do users write more insecure code with ai assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 2785–2799, New York, NY, USA, 2023. Association for Computing Machinery.

[P21] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 491–514, 2023.

[P22] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*, pages 1–7, 2022.

[P23] Ramazan Yilmaz and Fatma Gizem Karaoglan Yilmaz. Augmented intelligence in programming learning: Examining student views on the use of chatgpt for programming learning. *Computers in Human Behavior: Artificial Humans*, 1(2):100005, 2023.

[P24] Mateusz Jaworski and Dariusz Piotrkowski. Study of software developers' experience using the github copilot tool in the software development process. *arXiv preprint arXiv:2301.04991*, 2023.

[P25] Cynthia Zastudil, Magdalena Rogalska, Christine Kapp, Jennifer Vaughn, and Stephen MacNeil. Generative ai in computing education: Perspectives of students and instructors. In *2023 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE, 2023.

[P26] Abdulhadi Shoufan. Exploring students' perceptions of chatgpt: Thematic analysis and follow-up survey. *IEEE Access*, 11:38805–38818, 2023.

[P27] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. Teaching cs50 with ai: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 750–756, 2024.

[P28] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. Self-regulation, self-efficacy, and fear of failure interactions with how novices use llms to solve programming problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2024, page 276–282, New York, NY, USA, 2024. Association for Computing Machinery.

[P29] Yuankai Xue, Hanlin Chen, Gina R Bai, Robert Tairas, and Yu Huang. Does chatgpt help with introductory programming? an experiment of students using chatgpt in cs1. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*, pages 331–341, 2024.

[P30] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[P31] Hauke Sandhaus, Maria Teresa Parreira, and Wendy Ju. Student reflections on self-initiated genai use in hci education. In *CHI '24 Workshop: LLMs as Research Tools: Applications and Evaluations in HCI Data Work*, 2024. Workshop Paper.

[P32] Ebtesam Al Haque, Chris Brown, Thomas D. LaToza, and Brittany Johnson. The evolution of information seeking in software development: Understanding the role and impact of ai assistants. In *Proceedings of the Human-Centered AI for Software Engineering (HumanAISE) Workshop at the 33rd ACM International Conference on the Foundations of Software Engineering (FSE 2025)*, 2025. To appear.

[P33] Irene Hou, Sophia Mettille, Owen Man, Zhuo Li, Cynthia Zastudil, and Stephen MacNeil. The effects of generative ai on computing students' help-seeking preferences. In *Proceedings of the 26th Australasian Computing Education Conference*, pages 39–48, 2024.

[P34] Philipp Haindl and Gerald Weinberger. Students' experiences of using chatgpt in an undergraduate programming course. *IEEE Access*, 12:43519–43529, 2024.

[P35] Mircea-Serban Vasiliniuc and Adrian Groza. Case study: using ai-assisted code generation in mobile teams. In *2023 IEEE 19th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 339–346. IEEE, 2023.

[P36] Stephanie Yang, Hanzhang Zhao, Yudian Xu, Karen Brennan, and Bertrand Schneider. Debugging with an ai tutor: Investigating novice help-seeking behaviors and perceived learning. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*, pages 84–94, 2024.

[P37] Hieke Keuning, Isaac Alpizar-Chacon, Ioanna Lykourentzou, Lauren Beehler, Christian Köppe, Imke de Jong, and Sergey Sosnovsky. Students' perceptions and use of generative ai tools for programming across different computing courses. In *Proceedings of the 24th Koli Calling International Conference on Computing Education Research*, pages 1–12, 2024.

[P38] Muhammad Waseem, Teerath Das, Aakash Ahmad, Peng Liang, Mahdi Fahmideh, and Tommi Mikkonen. Chatgpt as a software development bot: A project-based study. In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*, pages 406–413. INSTICC, SciTePress, 2024.

[P39] Ze Shi Li, Nowshin Nawar Arony, Ahmed Musa Awon, Daniela Damian, and Bowen Xu. Ai tool use and adoption in software development by individuals and organizations: A grounded theory study. *arXiv preprint arXiv:2406.17325*, 2024.

[P40] Mounika Yabaku and Sofia Ouhbi. University students' perception and expectations of generative ai tools for software engineering. In *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–5. IEEE, 2024.

[P41] Samuel Boguslawski, Rowan Deer, and Mark G Dawson. Programming education and learner motivation in the age of generative ai: student and educator perspectives. *Information and Learning Sciences*, 2024.

[P42] Xin Tan, Xiao Long, Xianjun Ni, Yinghao Zhu, Jing Jiang, and Li Zhang. How far are ai-powered programming assistants from meeting developers' needs? *arXiv preprint arXiv:2404.12000*, 2024.

[P43] Agnia Sergeyuk, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. Using ai-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology*, 178:107610, 2025.

[P44] Albina Shynkar. Investigating the influence of code generating technologies on learning process of novice programmers in higher education computer science course. In *Proceedings of the 39th Twente Student Conference on IT*, Enschede, The Netherlands, 2023. University of Twente. Bachelor thesis.

[P45] Nicolay Johansen and Fredric Mourath. Generative ai in the gaming realm a developer survey, 2024.

[P46] Rudrajit Choudhuri, Dylan Liu, Igor Steinmacher, Marco Gerosa, and Anita Sarma. How far are we? the triumphs and trials of generative ai in learning software engineering. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[P47] Itzhak Aviv, Moshe Leiba, Havana Rika, and Yogev Shani. The impact of chatgpt on students' learning programming languages. In *International Conference on Human-Computer Interaction*, pages 207–219. Springer, 2024.

[P48] Rudaiba Adnin, Atharva Pandkar, Bingsheng Yao, Dakuo Wang, and Maitraye Das. Examining student and teacher perspectives on undisclosed use of generative ai in academic work. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery.

[P49] Mireilla Bikanga Ada. It helps with crap lecturers and their low effort: Investigating computer science students' perceptions of using chatgpt for learning. *Education Sciences*, 14(10):1106, 2024.

[P50] Anshul Shah, Anya Chernova, Elena Tomson, Leo Porter, William G Griswold, and Adalbert Gerald Soosai Raj. Students' use of github copilot for working with large code bases. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 1050–1056, 2025.

[P51] Valeria Ramirez Osorio, Angela Zavaleta Bernuy, Bogdan Simion, and Michael Liut. Understanding the impact of using generative ai tools in a database course. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 959–965, 2025.

[P52] Deo Shao and Fredrick Ishengoma. Empirical analysis of generative ai adoption in software development. *Available at SSRN 5149425*, 2025.

[P53] Benyamin Tabarsi, Heidi Reichert, Ally Limke, Sandeep Kuttal, and Tiffany Barnes. Llms' reshaping of people, processes, products, and society in software development: A comprehensive exploration with early adopters. *arXiv preprint arXiv:2503.05012*, 2025.

[P54] Rina Zviel-Girshin. The good and bad of ai tools in novice programming education. *Education Sciences*, 14(10):1089, 2024.

[P55] Arina Kudriavtseva, Nisar Ahmad Hotak, and Olga 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 40th ACM/SIGAPP Symposium on Applied Computing*, pages 1637–1646, 2025.

[P56] Alina Mailach, Dominik Gorgosch, Norbert Siegmund, and Janet Siegmund. "ok pal, we have to code that now": interaction patterns of programming beginners with a conversational chatbot. *Empirical Software Engineering*, 30(1):34, 2025.

[P57] Eric D Manley, Timothy Urness, Andrei Migunov, and Md Alimoor Reza. Examining student use of ai in cs1 and cs2. *Journal of Computing Sciences in Colleges*, 39(6):41–51, 2024.

[P58] Seongyune Choi and Hyeoncheol Kim. The impact of a large language model-based programming learning environment on students' motivation and programming ability. *Education and Information Technologies*, pages 1–30, 2024.

[P59] Jamie Gorson Benario, Jenn Marroquin, Monica M Chan, Ernest DV Holmes, and Daniel Mejia. Unlocking potential with generative ai instruction: Investigating mid-level software development student perceptions, behavior, and adoption. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, pages 395–401, 2025.

[P60] Christian Rahe and Walid Maalej. How do programming students use generative ai? *Proceedings of the ACM on Software Engineering*, 2(FSE):978–1000, 2025.

[P61] Isaac Alpizar-Chacon and Hieke Keuning. Student's use of generative ai as a support tool in an advanced web development course. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 312–318, 2025.

[P62] Rudrajit Choudhuri, Ambareesh Ramakrishnan, Amreeta Chatterjee, Bianca Trinkenreich, Igor Steinmacher, Marco Gerosa, and Anita Sarma. Insights from the frontline: Genai utilization among software engineering students. In *2025 IEEE/ACM 37th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–12. IEEE, 2025.

[P63] Uwe M Borghoff, Mark Minas, and Jannis Schopp. Generative ai in student software development projects: A user study on experiences and self-assessment. In *Proceedings of the 6th European Conference on Software Engineering Education*, pages 161–170, 2025.

[P64] Christian Jay St Francis Clarke and Abdullah Konak. The impact of ai use in programming courses on critical thinking skills. *Journal of Cybersecurity Education, Research and Practice*, 2025(1):5, 2025.

[P65] Sanka Rasnayaka, Guanlin Wang, Ridwan Shariffdeen, and Ganesh Neelakanta Iyer. An empirical study on usage and perceptions of llms in a software engineering project. In *Proceedings of the 1st International Workshop on Large Language Models for Code*, pages 111–118, 2024.

[P66] Bruno Pereira Cipriano and Pedro Alves. " chatgpt is here to help, not to replace anybody"–an evaluation of students' opinions on integrating chatgpt in cs courses. *arXiv preprint arXiv:2404.17443*, 2024.

[P67] Pedro Alves and Bruno Pereira Cipriano. "give me the code"–log analysis of first-year cs students' interactions with gpt. *arXiv preprint arXiv:2411.17855*, 2024.

[P68] Wendy Mendes, Samara Souza, and Cleidson De Souza. "you're on a bicycle with a little motor": Benefits and challenges of using ai code assistants. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering*, pages 144–152, 2024.

[P69] Mehmet Akhoroz and Caglar Yildirim. Conversational ai as a coding assistant: Understanding programmers' interactions with and expectations from large language models for coding. *arXiv preprint arXiv:2503.16508*, 2025.

[P70] Aadarsh Padiyath, Xinying Hou, Amy Pang, Diego Viramontes Vargas, Xingjian Gu, Tamara Nelson-Fromm, Zihan Wu, Mark Guzdial, and Barbara Ericson. Insights from social shaping theory: The appropriation of large language models in an undergraduate programming course. In *Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1*, pages 114–130, 2024.

[P71] Marina Lepp and Joosep Kaimre. Does generative ai help in learning programming: Students' perceptions, reported use and relation to performance. *Computers in Human Behavior Reports*, 18:100642, 2025.

[P72] Irene Hou, Owen Man, Kate Hamilton, Srishty Muthusekaran, Jeffin Johnykutty, Leili Zadeh, and Stephen MacNeil. 'all roads lead to chatgpt': How generative ai is eroding social interactions and student learning communities. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 79–85, 2025.

[P73] Kai Korpimies, Antti Laaksonen, and Matti Luukkainen. Unrestricted use of llms in a software project course: Student perceptions on learning and impact on course performance. In *Proceedings of the 24th Koli Calling International Conference on Computing Education Research*, pages 1–7, 2024.

[P74] Amy Pang, Aadarsh Padiyath, Diego Viramontes Vargas, and Barbara Jane Ericson. Examining the relationship between socioeconomic status and beliefs about large language models in an undergraduate programming course. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 1*, pages 172–178, 2024.

[P75] Aashish Ghimire and John Edwards. Coding with ai: How are tools like chatgpt being used by students in foundational programming courses. In *International Conference on Artificial Intelligence in Education*, pages 259–267. Springer, 2024.

[P76] Kehinde Aruleba, Ismaila Temitayo Sanusi, George Obaido, and Blessing Ogbuokiri. Integrating chatgpt in a computer science course: Students perceptions and suggestions. *arXiv preprint arXiv:2402.01640*, 2023.

[P77] Abdessalam Ouaazki, Kristoffer Bergram, Juan Carlos Farah, Denis Gillet, and Adrian Holzer. Generative ai-enabled conversational interaction to support self-directed learning experiences in transversal computational thinking. In *Proceedings of the 6th ACM Conference on Conversational User Interfaces*, pages 1–12, 2024.

[P78] Wenhan Lyu, Yimeng Wang, Yifan Sun, and Yixuan Zhang. Will your next pair programming partner be human? an empirical evaluation of generative ai as a collaborative teammate in a semester-long classroom setting. *arXiv preprint arXiv:2505.08119*, 2025.

[P79] Mario Simaremare, Chandro Pardede, Irma Tampubolon, Putri Manurung, and Daniel Simangunsong. Pair programming in programming courses in the era of generative ai: Students' perspective. In *2024 31st Asia-Pacific Software Engineering Conference (APSEC)*, pages 507–511. IEEE, 2024.

[P80] Adam Alami and Neil Ernst. Human and machine: How software engineers perceive and engage with ai-assisted code reviews compared to their peers. In *2025 IEEE/ACM 18th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 63–74. IEEE, 2025.

References

[1] A. Frank Ackerman, Lynne S. Buchwald, and Frank H. Lewski. 1989. Software inspections: an effective verification process. *IEEE software* 6, 3 (1989), 31–36.

[2] A Frank Ackerman, Priscilla J Fowler, and Robert G Ebenau. 1984. Software inspections and the industrial production of software. In *Proc. of a symposium on Software validation: inspection-testing-verification-alternatives*. 13–40.

[3] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. 2020. Software documentation: the practitioners' perspective. In *Proceedings of the acm/ieee 42nd international conference on software engineering*. 590–601.

[4] Muhammad Ovais Ahmad, Iftikhar Ahmad, and Fawad Qayum. 2024. Early career software developers and work preferences in software engineering. *Journal of Software: Evolution and Process* 36, 2 (2024), e2513.

[5] Toufique Ahmed, Christian Bird, Premkumar Devanbu, and Saikat Chakraborty. 2024. Studying llm performance on closed-and open-source data. *arXiv preprint arXiv:2402.15100* (2024).

[6] Kamel Alboaouh. 2018. The gap between engineering schools and industry: A strategic initiative. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–6.

[7] Apostolos Ampatzoglou, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. 2019. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology* 106 (2019), 201–230.

[8] Anonymous. 2025. Hot take: Vibe Coding is NOT the future. https://www.reddit.com/r/ChatGPTCoding/comments/1iueymf/hot_take_vibe_coding_is_not_the_future/. Online forum post. Reddit. Retrieved July 18, 2025.

[9] Anonymous. 2025. Read a software engineering blog if you think you like coding. https://www.reddit.com/r/vibecoding/comments/1kprxpl/read_a_software_engineering_blog_if_you_think/. [Online forum post]. Reddit. Retrieved July 18, 2025.

[10] Kitchenham BA and Stuart Charters. 2007. Guidelines for performing systematic literature reviews in software engineering. *Vol. 2, Jan* (2007).

[11] Sebastian Baltes and Stephan Diehl. 2018. Towards a theory of software development expertise. In *Proceedings of the 2018 26th acm joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 187–200.

[12] Victor R Basili and Richard W Selby. 2006. Comparing the effectiveness of software testing strategies. *IEEE transactions on software engineering* 12 (2006), 1278–1296.

[13] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 500–506.

[14] Brett A Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. 2019. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *Proceedings of the working group reports on innovation and technology in computer science education* (2019), 177–210.

[15] Andrew Begel and Beth Simon. 2008. Novice software developers, all over again. In *Proceedings of the fourth international workshop on computing education research*. 3–14.

[16] Brian Bell, Teresa Thomas, Sang Won Lee, and Chris Brown. 2025. How do Software Engineering Candidates Prepare for Technical Interviews? *arXiv preprint arXiv:2507.02068* (2025).

[17] Jose Berengueres. 2024. How to regulate large language models for responsible AI. *IEEE Transactions on Technology and Society* 5, 2 (2024), 191–197.

[18] Roberta MM Bezerra, Fabio QB da Silva, Anderson M Santana, Cleyton VC Magalhaes, and Ronnie ES Santos. 2015. Replication of empirical studies in software engineering: An update of a systematic mapping study. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–4.

[19] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. 2011. Don't touch my code! Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 4–14.

[20] Eva Alice Christiane Bittner and Jan Marco Leimeister. 2013. Why shared understanding matters–Engineering a collaboration process for shared understanding to improve collaboration effectiveness in heterogeneous teams. In *2013 46th Hawaii international conference on system sciences*. IEEE, 106–114.

[21] Jeanette Blomberg and Mark Burrel. 2009. An ethnographic approach to design. In *Human-computer interaction*. CRC Press, 87–110.

[22] Jeanette Blomberg and Helena Karasti. 2013. Reflections on 25 years of ethnography in CSCW. *Computer supported cooperative work (CSCW)* 22, 4 (2013), 373–423.

[23] Jessica Y Bo, Majeed Kazemitabaar, Emma Zhuang, and Ashton Anderson. 2025. Who's the Leader? Analyzing Novice Workflows in LLM-Assisted Debugging of Machine Learning Code. *arXiv preprint arXiv:2505.08063* (2025).

[24] Joel Brandt, Philip J Guo, Joel Lewenstein, Mira Dontcheva, and Scott R Klemmer. 2009. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 1589–1598.

[25] Beatriz Cabrero-Daniel, Yasamin Fazelidehkordi, and Ali Nouri. 2024. How Can Generative AI Enhance Software Management? Is It Better Done than Perfect? In *Generative AI for Effective Software Development*. Springer, 235–255.

[26] Doga Cambaz and Xiaoling Zhang. 2024. Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 172–178.

[27] Edna Dias Canedo, Angélica Toffano Seidel Calazans, Geovana Ramos Sousa Silva, Pedro Henrique Teixeira Costa, Rodrigo Pereira de Mesquita, and Eloisa Toffano Seidel Masson. 2022. Creativity and design thinking as facilitators in requirements elicitation. *International Journal of Software Engineering and Knowledge Engineering* 32, 10 (2022), 1527–1558.

[28] Rafael Capilla, Olaf Zimmermann, Carlos Carrillo, and Hernán Astudillo. 2020. Teaching students software architecture decision making. In *Software Architecture: 14th European Conference, ECSA 2020, L'Aquila, Italy, September 14–18, 2020, Proceedings 14*. Springer, 231–246.

[29] Diego Cedrim, Alessandro Garcia, Melina Mongiovi, Rohit Gheyi, Leonardo Sousa, Rafael De Mello, Baldoino Fonseca, Márcio Ribeiro, and Alexander Chávez. 2017. Understanding the impact of refactoring on smells: A longitudinal study of 23 software projects. In *Proceedings of the 2017 11th Joint Meeting on foundations of Software Engineering*. 465–475.

[30] Preetha Chatterjee, Minji Kong, and Lori Pollock. 2020. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. *Journal of Systems and Software* 159 (2020), 110454.

[31] Alyssia Chen, Carol Wong, Katy Tarrit, and Anthony Peruma. 2024. Impostor Syndrome in Final Year Computer Science Students: An Eye Tracking and Biometrics Study. In *International Conference on Human-Computer Interaction*. Springer, 22–41.

[32] Chunyang Chen and Zhenchang Xing. 2016. Towards correlating search on google and asking on stack overflow. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. IEEE, 83–92.

[33] Junkai Chen, Xing Hu, Zhenhao Li, Cuiyun Gao, Xin Xia, and David Lo. 2024. Code search is all you need? improving code suggestions with code search. In *Proceedings of the IEEE/ACM 46th international conference on software engineering*. 1–13.

[34] Pauline Rose Clance and Suzanne Ament Imes. 1978. The imposter phenomenon in high achieving women: Dynamics and therapeutic intervention. *Psychotherapy: Theory, research & practice* 15, 3 (1978), 241.

[35] Michelle Craig, Phill Conrad, Dylan Lynch, Natasha Lee, and Laura Anthony. 2018. Listening to early career software developers. *J. Comput. Sci. Coll.* 33, 4 (April 2018), 138–149.

[36] Broderick Crawford, Ricardo Soto, Claudio León de la Barra, Kathleen Crawford, and Eduardo Olguín. 2014. The influence of emotions on productivity in software engineering. In *International Conference on Human-Computer Interaction*. Springer, 307–310.

[37] Zheyuan Kevin Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2024. The effects of generative ai on high skilled work: Evidence from three field experiments with software developers. *Available at SSRN 4945566* (2024).

[38] Fabio QB Da Silva, Marcos Suassuna, A César C França, Alicia M Grubb, Tatiana B Gouveia, Cleviton VF Monteiro, and Igor Ebrahim dos Santos. 2014. Replication of empirical studies in software engineering research: a systematic mapping study. *Empirical Software Engineering* 19, 3 (2014), 501–557.

[39] Barthélémy Dagenais, Harold Ossher, Rachel KE Bellamy, Martin P Robillard, and Jacqueline P De Vries. 2010. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 275–284.

[40] Gabriele De Seta, Matti Pohjonen, and Aleksi Knuutila. 2024. Synthetic ethnography: Field devices for the qualitative study of generative models. *Big Data & Society* 11, 4 (2024), 20539517241303126.

[41] Derek Debellis, Kevin Storer, Daniella Villalba, Nathen Harvey, Sarah D'Angelo, and Adam Brown. 2024. *Superagency in the workplace: Empowering people to unlock AI's full potential*. https://dora.dev/research/ai/gen-ai-report/

[42] Zehang Deng, Wanlun Ma, Qing-Long Han, Wei Zhou, Xiaogang Zhu, Sheng Wen, and Yang Xiang. 2025. Exploring DeepSeek: A Survey on Advances, Applications, Challenges and Future Directions. *IEEE/CAA Journal of Automatica Sinica* 12, 5 (2025), 872–893.

[43] Yogesh K Dwivedi, Nir Kshetri, Laurie Hughes, Emma Louise Slade, Anand Jeyaraj, Arpan Kumar Kar, Abdullah M Baabdullah, Alex Koohang, Vishnupriya Raghavan, Manju Ahuja, et al. 2023. Opinion Paper:"So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. *International Journal of Information Management* 71 (2023), 102642.

[44] Tore Dybå and Torgeir Dingsøyr. 2008. Empirical studies of agile software development: A systematic review. *Information and software technology* 50, 9-10 (2008), 833–859.

[45] Christof Ebert and Panos Louridas. 2023. Generative AI for software practitioners. *IEEE Software* 40, 4 (2023), 30–38.

[46] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 31–53.

[47] Mark Fenwick, Paul Jurcys, and Valto Loikkanen. 2024. Welcome to Google Zero! Incentives, Remuneration & Copyright in a New World of AI Agents. *Incentives, Remuneration & Copyright in a New World of AI Agents (May 27, 2024)* (2024).

[48] Samuel Ferino, Rashina Hoda, John Grundy, and Christoph Treude. 2025. Supplementary Information Package - SLR. https://github.com/Samuellucas97/SupplementaryInfoPackage-SLR

[49] Samuel Ferino, Rashina Hoda, John Grundy, and Christoph Treude. 2025. Walking the Tightrope of LLMs for Software Development: A Practitioners' Perspective. *arXiv preprint arXiv:2511.06428* (2025).

[50] Brian Fitzgerald and Tom O'Kane. 1999. A longitudinal study of software process improvement. *IEEE software* 16, 3 (1999), 37–45.

[51] Stephen L France. 2024. Navigating software development in the ChatGPT and GitHub Copilot era. *Business Horizons* 67, 5 (2024), 649–661.

[52] Fabian Gilson, Miguel Morales-Trujillo, and Moffat Mathews. 2020. How junior developers deal with their technical debt?. In *Proceedings of the 3rd International Conference on Technical Debt*. 51–61.

[53] Muhammet Kürşat Görmez, Murat Yılmaz, and Paul M Clarke. 2024. Large Language Models for Software Engineering: A Systematic Mapping Study. In *European Conference on Software Process Improvement*. Springer, 64–79.

[54] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2017. Unhappy developers: Bad for themselves, bad for process, and bad for software product. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 362–364.

[55] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2018. What happens when software developers are (un) happy. *Journal of Systems and Software* 140 (2018), 32–47.

[56] Paloma Guenes, Rafael Tomaz, Marcos Kalinowski, Maria Teresa Baldassarre, and Margaret-Anne Storey. 2024. Impostor phenomenon in software engineers. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*. 96–106.

[57] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196* (2024).

[58] Shalini Harkar. 2025. What is Vibe Coding? https://www.ibm.com/think/topics/vibe-coding. Accessed: 2025-07-28.

[59] Junda He, Christoph Treude, and David Lo. 2025. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–30.

[60] Dulaji Hidellaarachchi, John Grundy, Rashina Hoda, and Kashumi Madampe. 2021. The effects of human aspects on the requirements engineering process: A systematic literature review. *IEEE Transactions on Software Engineering* 48, 6 (2021), 2105–2127.

[61] Rashina Hoda, Norsaremah Salleh, John Grundy, and Hui Mien Tee. 2017. Systematic literature reviews in agile software development: A tertiary study. *Information and software technology* 85 (2017), 60–70.

[62] Lars Erik Holmquist and Sam Nemeth. 2025. "Don't believe anything I tell you, it's all lies!": A Synthetic Ethnography on Untruth in Large Language Models. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–6.

[63] Andre Hora. 2021. Googling for software development: What developers search for and what they find. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 317–328.

[64] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* (Sept. 2024). https://doi.org/10.1145/3695988

[65] Fadhl Hujainah, Rohani Binti Abu Bakar, Mansoor Abdullateef Abdulgabber, and Kamal Z Zamli. 2018. Software requirements prioritisation: a systematic literature review on significance, stakeholders, techniques and challenges. *IEEE Access* 6 (2018), 71497–71523.

[66] Mia Mohammad Imran, Preetha Chatterjee, and Kostadin Damevski. 2024. Uncovering the causes of emotions in software developer communication using zero-shot llms. In *Proceedings of the IEEE/ACM 46th international conference on software engineering*. 1–13.

[67] Kong Inc. 2025. *Generative AI Enterprise Trends Report 2025*. https://konghq.com/resources/reports/generative-ai-enterprise-trends-2025 Accessed: 2025-07-04.

[68] Victoria Jackson, Bogdan Vasilescu, Daniel Russo, Paul Ralph, Rafael Prikladnicki, Maliheh Izadi, Sarah D'angelo, Sarah Inman, Anielle Andrade, and André Van Der Hoek. 2025. The impact of generative AI on creativity in software development: A research agenda. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–28.

[69] Magne Jørgensen, Gunnar Rye Bergersen, and Knut Liestøl. 2020. Relations between effort estimates, skill indicators, and measured programming skill. *IEEE Transactions on Software Engineering* 47, 12 (2020), 2892–2906.

[70] Magne Jørgensen and Dag IK Sjøberg. 2001. Impact of effort estimates on software project work. *Information and software technology* 43, 15 (2001), 939–948.

[71] Tobias Kaatz. 2014. Hiring in the software industry. *IEEE Software* 31, 6 (2014), 96–96.

[72] Aymen Kabir, Suraj Shah, Alexander Haddad, and Daniel MS Raper. 2025. Introducing our custom GPT: An example of the potential impact of personalized GPT builders on scientific writing. *World Neurosurgery* 193 (2025), 461–468.

[73] Matthew Kam, Cody Miller, Miaoxin Wang, Abey Tidwell, Irene A Lee, Joyce Malyn-Smith, Beatriz Perez, Vikram Tiwari, Joshua Kenitzer, Andrew Macvean, et al. 2025. What do professional software developers need to know to succeed in an age of Artificial Intelligence? *arXiv preprint arXiv:2506.00202* (2025).

[74] Amanpreet Kapoor and Christina Gardner-McCune. 2019. Understanding CS undergraduate students' professional development through the lens of internship experiences. In *Proceedings of the 50th ACM technical symposium on computer science education*. 852–858.

[75] Amanpreet Kapoor and Christina Gardner-McCune. 2020. Exploring the participation of CS undergraduate students in industry internships. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1103–1109.

[76] Kai-Kristian Kemell, Matti Saarikallio, Anh Nguyen-Duc, and Pekka Abrahamsson. 2025. Still just personal assistants?–A multiple case study of generative AI adoption in software organizations. *Information and Software Technology* (2025), 107805.

[77] Chris F. Kemerer and Sandra Slaughter. 2002. An empirical approach to studying software evolution. *IEEE transactions on software engineering* 25, 4 (2002), 493–509.

[78] Hourieh Khalajzadeh and John Grundy. 2024. Accessibility of low-code approaches: A systematic literature review. *Information and Software Technology* (2024), 107570.

[79] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. 2009. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology* 51, 1 (2009), 7–15.

[80] Barbara Kitchenham, Lech Madeyski, and David Budgen. 2022. SEGRESS: Software engineering guidelines for reporting secondary studies. *IEEE Transactions on Software Engineering* 49, 3 (2022), 1273–1298.

[81] Barbara Kitchenham, Rialette Pretorius, David Budgen, O Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. 2010. Systematic literature reviews in software engineering–a tertiary study. *Information and software technology* 52, 8 (2010), 792–805.

[82] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: How developers see it. In *Proceedings of the 38th international conference on software engineering*. 1028–1038.

[83] Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Beresnitzky, Iris Braunstein, and Pattie Maes. 2025. Your brain on chatgpt: Accumulation of cognitive debt when using an ai assistant for essay writing task. *arXiv preprint arXiv:2506.08872* (2025).

[84] Elvan Kula, Eric Greuter, Arie Van Deursen, and Georgios Gousios. 2021. Factors affecting on-time delivery in large-scale agile software development. *IEEE Transactions on Software Engineering* 48, 9 (2021), 3573–3592.

[85] Harsh Kumar, Jonathan Vincentius, Ewan Jordan, and Ashton Anderson. 2025. Human creativity in the age of llms: Randomized experiments on divergent and convergent thinking. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–18.

[86] Miikka Kuutila, Leevi Rantala, Junhao Li, Simo Hosio, and Mika Mäntylä. 2024. What Makes Programmers Laugh? Exploring the Submissions of the Subreddit r/ProgrammerHumor.. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 371–381.

[87] Viet Dac Lai, Nghia Trung Ngo, Amir Pouran Ben Veyseh, Hieu Man, Franck Dernoncourt, Trung Bui, and Thien Huu Nguyen. 2023. Chatgpt beyond english: Towards a comprehensive evaluation of large language models in multilingual learning. *arXiv preprint arXiv:2304.05613* (2023).

[88] Thomas D LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*. 492–501.

[89] Lucas Layman, Madeline Diep, Meiyappan Nagappan, Janice Singer, Robert Deline, and Gina Venolia. 2013. Debugging revisited: Toward understanding the debugging needs of contemporary software developers. In *2013 ACM/IEEE international symposium on empirical software engineering and measurement*. IEEE, 383–392.

[90] Hao-Ping Lee, Advait Sarkar, Lev Tankelevitch, Ian Drosos, Sean Rintel, Richard Banks, and Nicholas Wilson. 2025. The impact of generative AI on critical thinking: Self-reported reductions in cognitive effort and confidence effects from a survey of knowledge workers. In *Proceedings of the 2025 CHI conference on human factors in computing systems*. 1–22.

[91] Valentina Lenarduzzi, Vladimir Mandić, Andrej Katin, and Davide Taibi. 2020. How long do junior developers take to remove technical debt items?. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–6.

[92] Annie Li, Madeline Endres, and Westley Weimer. 2022. Debugging with stack overflow: Web search behavior in novice and expert programmers. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*. 69–81.

[93] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. 2013. What help do developers seek, when and how?. In *2013 20th working conference on reverse engineering (WCRE)*. IEEE, 142–151.

[94] Jenny T Liang, Melissa Lin, Nikitha Rao, and Brad A Myers. 2025. Prompts are programs too! understanding how developers build software containing prompts. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 1591–1614.

[95] Dastyni Loksa and Amy J Ko. 2016. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM conference on international computing education research*. 83–91.

[96] Yacine Majdoub and Eya Ben Charrada. 2024. Debugging with open-source large language models: An evaluation. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 510–516.

[97] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. 2011. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 2857–2866.

[98] Zainab Masood, Rashina Hoda, Kelly Blincoe, and Daniela Damian. 2022. Like, dislike, or just do it? How developers approach software development tasks. *Information and Software Technology* 150 (2022), 106963.

[99] Hannah Mayer, L Yee, M Chui, and R Roberts. 2025. Superagency in the workplace: Empowering people to unlock AI's full potential. *McKinsey Digital* 28 (2025).

[100] Fairuz Nawer Meem. 2024. Effective Integration and Use of Non-Development LLMs in Software Development. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 374–375.

[101] Imdad Ahmad Mian, Aamir Anwar, Roobaea Alroobaea, Syed Sajid Ullah, Fahad Almansour, and Fazlullah Umar. 2022. A comprehensive skills analysis of novice software developers working in the professional software development industry. *Complexity* 2022, 1 (2022), 2631727.

[102] Harlan D Mills. 1980. The management of software engineering, Part I: Principles of software engineering. *IBM Systems Journal* 19, 4 (1980), 414–420.

[103] Mia Minnes, Sheena Ghanbari Serslev, and Omar Padilla. 2021. What do cs students value in industry internships? *ACM Transactions on Computing Education (TOCE)* 21, 1 (2021), 1–15.

[104] Rahul Mohanani, Prabhat Ram, Ahmed Lasisi, Paul Ralph, and Burak Turhan. 2017. Perceptions of creativity in software engineering research and practice. In *2017 43rd euromicro conference on software engineering and advanced applications (seaa)*. IEEE, 210–217.

[105] Arthur-Jozsef Molnar, Simona Motogna, Diana Cristea, and Diana-Florina Sotropa. 2024. Exploring Complexity Issues in Junior Developer Code Using Static Analysis and FCA. In *2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 407–414.

[106] Kjetil Molokken and Magne Jorgensen. 2003. A review of software surveys on software effort estimation. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings*. IEEE, 223–230.

[107] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th working conference on mining software repositories*. 262–271.

[108] Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. 2011. How we refactor, and how we know it. *IEEE Transactions on Software Engineering* 38, 1 (2011), 5–18.

[109] Hira Naveed, Chetan Arora, Hourieh Khalajzadeh, John Grundy, and Omar Haggag. 2024. Model driven engineering for machine learning components: A systematic literature review. *Information and Software Technology* (2024), 107423.

[110] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 1–5.

[111] Haoran Niu, Iman Keivanloo, and Ying Zou. 2017. Learning to rank code examples for code search engines. *Empirical Software Engineering* 22, 1 (2017), 259–291.

[112] Anu Niva, Jouni Markkula, and Elina Annanperä. 2023. Junior software engineers' international communication and collaboration competences. *IEEE Access* 11 (2023), 139039–139068.

[113] Damla Oguz and Kaya Oguz. 2019. Perspectives on the gap between the software industry and the software engineering education. *Ieee Access* 7 (2019), 117527–117543.

[114] Juanan Pereira, Juan-Miguel López, Xabier Garmendia, and Maider Azanza. 2024. Leveraging open source LLMs for software engineering education and training. In *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 1–10.

[115] Farman Ali Pirzado, Awais Ahmed, Román A Mendoza-Urdiales, and Hugo Terashima-Marin. 2024. Navigating the pitfalls: Analyzing the behavior of LLMs as a coding assistant for computer science students-a systematic review of the literature. *IEEE Access* (2024).

[116] Sebastián Pizard, Diego Vallespir, and Barbara Kitchenham. 2022. A longitudinal case study on the effects of an evidence-based software engineering training. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*. 1–13.

[117] James Prather, Juho Leinonen, Natalie Kiesler, Jamie Gorson Benario, Sam Lau, Stephen MacNeil, Narges Norouzi, Simone Opel, Vee Pettit, Leo Porter, et al. 2025. Beyond the hype: A comprehensive review of current trends in generative AI research, teaching practices, and tools. *2024 Working Group Reports on Innovation and Technology in Computer Science Education* (2025), 300–338.

[118] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1–24.

[119] Nishat Raihan, Mohammed Latif Siddiq, Joanna C.S. Santos, and Marcos Zampieri. 2025. Large Language Models in Computer Science Education: A Systematic Literature Review. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 938–944. https://doi.org/10.1145/3641554.3701863

[120] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How do professional developers comprehend software?. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 255–265.

[121] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*. 181–190.

[122] Larissa Salerno, Christoph Treude, and Patanamon Thongtatunam. 2024. Open Source Software Development Tool Installation: Challenges and Strategies For Novice Developers. *arXiv preprint arXiv:2404.14637* (2024).

[123] Norsaremah Salleh, Emilia Mendes, and John Grundy. 2011. Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review. *IEEE Transactions on Software Engineering* 37, 4 (2011), 509–525. https://doi.org/10.1109/TSE.2010.59

[124] Advait Sarkar and Ian Drosos. 2025. Vibe coding: programming through conversation with artificial intelligence. *arXiv preprint arXiv:2506.23253* (2025).

[125] Yuya Sasaki, Hironori Washizaki, Jialong Li, Dominik Sander, Nobukazu Yoshioka, and Yoshiaki Fukazawa. 2024. Systematic Literature Review of Prompt Engineering Patterns in Software Engineering. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 670–675.

[126] Agnia Sergeyuk, Ilya Zakharov, Ekaterina Koshchenko, and Maliheh Izadi. 2025. Human-AI Experience in Integrated Development Environments: A Systematic Literature Review. *arXiv preprint arXiv:2503.06195* (2025).

[127] Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R Johnston, et al. 2023. Towards understanding sycophancy in language models. *arXiv preprint arXiv:2310.13548* (2023).

[128] Helen Sharp, Yvonne Dittrich, and Cleidson RB De Souza. 2016. The role of ethnographic studies in empirical software engineering. *IEEE Transactions on Software Engineering* 42, 8 (2016), 786–804.

[129] Jieke Shi, Zhou Yang, and David Lo. 2025. Efficient and Green Large Language Models for Software Engineering: Literature Review, Vision, and the Road Ahead. *ACM Trans. Softw. Eng. Methodol.* 34, 5, Article 137 (May 2025), 22 pages. https://doi.org/10.1145/3708525

[130] Patrick C Shih, Gina Venolia, and Gary M Olson. 2011. Brainstorming under constraints: why software developers brainstorm in groups. In *Proceedings of the 25th BCS conference on human-computer interaction*. 74–83.

[131] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why we refactor? confessions of GitHub contributors. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 858–870. https://doi.org/10.1145/2950290.2950305

[132] José Aldo Silva Da Costa and Rohit Gheyi. 2023. Evaluating the code comprehension of novices with eye tracking. In *Proceedings of the XXII Brazilian Symposium on Software Quality*. 332–341.

[133] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V Lopes. 2011. How well do search engines support code retrieval on the web? *ACM Transactions on Software Engineering and Methodology (TOSEM)* 21, 1 (2011), 1–25.

[134] Diane Strode, Torgeir Dingsøyr, and Yngve Lindsjorn. 2022. A teamwork effectiveness model for agile software development. *Empirical Software Engineering* 27, 2 (2022), 56.

[135] Senfa Sun. 2025. Addressing Impostor Phenomenon in Collaboration Between User Experience Designers and Software Developers. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '25)*. Association for Computing Machinery, New York, NY, USA, Article 64, 8 pages. https://doi.org/10.1145/3706599.3720009

[136] Yuan Sun and Ting Wang. 2025. Be friendly, not friends: How llm sycophancy shapes user trust. *arXiv preprint arXiv:2502.10844* (2025).

[137] Grace Taylor and Steven Clarke. 2022. A Tour Through Code: Helping Developers Become Familiar with Unfamiliar Code.. In *PPIG*. 114–126.

[138] Peeratham Techapalokul and Eli Tilevich. 2017. Novice programmers and software quality: Trends and implications. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 246–250.

[139] Timm Teubner, Christoph M Flath, Christof Weinhardt, Wil van der Aalst, and Oliver Hinz. 2023. Welcome to the era of chatgpt et al. the prospects of large language models. *Business & Information Systems Engineering* 65, 2 (2023), 95–101.

[140] David Thomas and Andrew Hunt. 2019. *The Pragmatic Programmer: your journey to mastery*. Addison-Wesley Professional.

[141] Claudia Tona, Reyes Juárez-Ramírez, Samantha Jiménez, and Mayra Durán. 2024. Exploring LLM Tools Through the Eyes of Industry Experts and Novice Programmers. In *2024 12th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE, 313–321.

[142] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* 50, 4 (2024), 911–936.

[143] Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S Yu, and Qingsong Wen. 2024. Large language models for education: A survey and outlook. *arXiv preprint arXiv:2403.18105* (2024).

[144] Emily Wenger and Yoed Kenett. 2025. We're Different, We're the Same: Creative Homogeneity Across LLMs. *arXiv preprint arXiv:2501.19361* (2025).

[145] Colin Werner, Ze Shi Li, Neil Ernst, and Daniela Damian. 2020. The lack of shared understanding of non-functional requirements in continuous software engineering: Accidental or essential?. In *2020 IEEE 28th international requirements engineering conference (RE)*. IEEE, 90–101.

[146] Jacqueline Whalley, Amber Settle, and Andrew Luxton-Reilly. 2023. A think-aloud study of novice debugging. *ACM Transactions on Computing Education* 23, 2 (2023), 1–38.

[147] James A Whittaker. 2002. What is software testing? And why is it so hard? *IEEE software* 17, 1 (2002), 70–79.

[148] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.

[149] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. 2012. *Experimentation in software engineering*. Vol. 236. Springer.

[150] Allison Woodruff, Renee Shelby, Patrick Gage Kelley, Steven Rousso-Schindler, Jamila Smith-Loud, and Lauren Wilcox. 2024. How knowledge workers think generative ai will (not) transform their industries. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–26.

[151] Michal R Wrobel. 2013. Emotions in the software development process. In *2013 6th International Conference on Human System Interactions (HSI)*. IEEE, 518–523.

[152] Fan Wu, Emily Black, and Varun Chandrasekaran. 2024. Generative monoculture in large language models. *arXiv preprint arXiv:2407.02209* (2024).

[153] Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. 2012. Developer prioritization in bug repositories. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 25–35.

[154] Zhou Yang, Jieke Shi, Prem Devanbu, and David Lo. 2024. Ecosystem of large language models for code. *ACM Transactions on Software Engineering and Methodology* (2024).

[155] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. 2024. A survey on large language models for software engineering. *arXiv preprint arXiv:2312.15223* (2024).

[156] Xiang Zhang, Senyu Li, Bradley Hauer, Ning Shi, and Grzegorz Kondrak. 2023. Don't trust ChatGPT when your question is not in English: a study of multilingual abilities and types of LLMs. *arXiv preprint arXiv:2305.16339* (2023).

[157] Xin Zhao and Narissa Tsuboi. 2024. Early career software developers-are you sinking or swimming?. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*. 166–176.

[158] Dewu Zheng, Yanlin Wang, Ensheng Shi, Hongyu Zhang, and Zibin Zheng. 2024. How well do llms generate code for different application domains? benchmark and evaluation. *arXiv preprint arXiv:2412.18573* (2024).

[159] Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. 2025. Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering* 30, 2 (2025), 50.

[160] Xiyu Zhou, Peng Liang, Beiqi Zhang, Zengyang Li, Aakash Ahmad, Mojtaba Shahin, and Muhammad Waseem. 2024. Exploring the problems, their causes and solutions of AI pair programming: A study on GitHub and stack overflow. *Journal of Systems and Software* (2024), 112204.

[161] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2024. Measuring GitHub Copilot's impact on productivity. *Commun. ACM* 67, 3 (2024), 54–63.