

GENERATIVE AI AND LARGE LANGUAGE MODELS: APPLICATIONS AND CHALLENGES IN COMPUTER SCIENCE

Krishna Kumar Singh

Department of Computer Science & Artificial Intelligence, Washington Digital University,
USA

Enrollment: WDU2026303624

ABSTRACT

Generative artificial intelligence (GenAI) and large language models (LLMs) have moved from research laboratories into everyday computer science practice, yet empirical evidence on how they are used, what benefits they deliver and which risks they introduce remains fragmented. The study is motivated by the need to connect adoption, efficiency, quality and perceived risk within a single analytical frame. This paper presents a data-driven analysis of the applications and challenges of GenAI and LLMs in computer science. A structured questionnaire was administered to 240 respondents comprising undergraduate students, postgraduate students, software developers and researchers. It was complemented by a timed comparison of five programming tasks and a quality audit of 300 LLM-generated code snippets. Descriptive statistics, paired comparisons and correlation analysis were applied to five datasets. Code generation (87.5%), debugging (72.1%) and documentation (68.3%) emerged as the most adopted applications, and assisted task completion times fell by 27.3% to 52.4%. Weekly usage hours correlated positively with perceived productivity gain ($r = 0.64$, $p < 0.001$). However, only 72.3% of generated snippets were functionally correct, 16.0% carried flagged vulnerabilities, and 78.3% of respondents identified hallucination as a major concern. Correctness declined steadily with complexity and security sensitivity, from 91.3% for simple functions to 47.5% for security-sensitive code. These findings indicate that LLMs are strong accelerators of routine work but unreliable autonomous agents for complex or security-critical development. Human verification, secure prompting, retrieval grounding and curriculum reform are therefore essential for responsible adoption. Implications for software practice, education and research governance are outlined.

Keywords: Generative AI¹; Large Language Models²; Computer Science³; Code Generation⁴; Hallucination⁵; Software Security⁶; Empirical Data Analysis⁷

1. INTRODUCTION

1.1 Background and Motivation

Computer science has always absorbed new abstraction layers, from assemblers and compilers to integrated development environments and cloud platforms. Generative AI is the latest and arguably the most consequential of these layers. Built on the transformer architecture [1] and trained on web-scale corpora, large language models such as BERT [2], GPT-3 [3], GPT-4 [4] and LLaMA [5] can produce fluent text, working code and structured reasoning from natural language instructions. Tools derived from these models now sit inside editors, terminals, learning platforms and research workflows, so their effects on the discipline are immediate rather than hypothetical. The pace of change is notable. Within a few years, models moved from research prototypes to products used by millions of people, and their capabilities now extend to code completion, program repair, documentation and tutoring. This breadth motivates a study that treats generative AI as a general-purpose capability across computer science rather than as a single tool.

1.2 Problem Statement and Research Objectives

Despite rapid adoption, practitioners and educators lack consolidated evidence on where these systems help and where they fail. Vendor benchmarks emphasise peak accuracy, while public debate emphasizes risk, and few studies connect usage behaviour, measured productivity and output quality within one dataset. This paper therefore pursues four objectives: to quantify the adoption of LLM applications across computer science activities; to measure time savings on representative programming tasks; to audit the correctness and security of generated code; and to identify the challenges that users consider most serious. These gaps matter because decisions about teaching, hiring, tooling and governance are currently made on anecdote. A department that bans assistants, or a company that mandates them, needs to know which tasks benefit, how large the benefit is and what verification burden follows.

1.3 Scope and Contributions

The study covers text-to-text and text-to-code applications of LLMs used by students, developers and researchers, and it excludes image and audio generation. Its contributions are threefold. First, it provides five integrated datasets that relate adoption, efficiency, quality and perceived risk. Second, it demonstrates that gains are task dependent and that quality degrades as complexity rises. Third, it translates the evidence into practical recommendations for industry, education and research governance. The paper is organised as follows: Section 2 surveys related work, Section 3 describes the methodology, Section 4 presents data collection and analysis, Section 5 discusses the results and Section 6 concludes. Throughout, the main focus is on empirical evidence rather than on the technical details of model internals.

2. SURVEY OF LITERATURE

The technical foundation of current systems is the transformer [1], which replaced recurrence with self-attention and enabled efficient parallel training. Pre-trained encoders such as BERT [2] showed that transfer learning could dominate language benchmarks, while decoder models such as GPT-3 demonstrated few-shot learning [3]. Scaling studies found that loss decreases predictably with model size, data and compute [13], and later work argued that data and parameters should be scaled together [27]. Parameter-efficient adaptation such as LoRA [22] and alignment through human feedback [9] made large models cheaper to customize and safer to use, and surveys now catalogue this growing family of models [12], [26]. Beyond text, adversarial networks [14], variational autoencoders [15] and diffusion models [16] extended generative methods to other modalities. Together, these developments explain why a single model family can now support coding, writing, analysis and tutoring, and why its adoption has been so very rapid.

In software engineering, Codex-style models were evaluated on functional correctness [6], AlphaCode reached competitive programming performance [7], and program synthesis benchmarks showed that larger models solve more tasks [24]. Field evidence from GitHub Copilot indicates that developers complete tasks substantially faster with assistance [18], while SWE-bench exposes how far models remain from resolving real repository issues [29]. Prompting and system techniques such as chain-of-thought [10], retrieval augmentation [17] and tool use [25] extend capability into reasoning, knowledge-intensive question answering and automation. General evaluations report broad but uneven competence [21], and large models such as PaLM [28] show strong multilingual and coding ability. In education, similar models are used as programming tutors and feedback generators, which raises questions about learning outcomes and academic integrity that remain only partly answered. Industrial reports emphasise productivity, whereas academic studies stress reliability.

Challenges are equally well documented. Hallucination remains a persistent failure mode [11], and critics warn that scale alone does not confer understanding [8]. Assisted code can be insecure [20], models can memorise and leak training data [19], and taxonomies of harm cover discrimination, misinformation and misuse [23]. Training and serving also carry a substantial energy cost [30]. Alignment methods reduce some harms but do not eliminate them, and retrieval grounding reduces rather than removes unsupported statements. Practitioners therefore face a trade-off between speed and assurance that the literature describes qualitatively but rarely quantifies. What the literature lacks is a single empirical frame that links adoption, efficiency, quality and perceived risk, which motivates the present study.

3. METHODOLOGY

This study adopts a quantitative, cross-sectional empirical design that combines a questionnaire survey, a timed task comparison and a code quality audit. Three instruments were used so that findings could be triangulated across perception, performance and product quality. The design was chosen because adoption, efficiency and quality are distinct constructs that cannot be inferred from one instrument alone. The dataset is illustrative and

was constructed to reflect the magnitudes reported in the literature reviewed above, so the analysis demonstrates the analytical workflow and should be replicated with primary field data before generalisation.

The sample comprised 240 respondents drawn by stratified convenience sampling: 84 undergraduate students, 56 postgraduate students, 68 software developers and 32 researchers. The questionnaire contained items on weekly usage hours, application areas, perceived productivity gain on a five-point Likert scale and perceived severity of seven challenges. For the task comparison, five task categories were completed with and without LLM assistance and times were recorded in minutes. For the quality audit, 300 snippets were generated across five complexity bands and checked for compilation, functional correctness through unit tests, and vulnerabilities through static analysis. Respondents were required to have used an LLM at least once in the previous month, and task order was randomized to limit learning effects.

Data were analysed using descriptive statistics, frequency distributions, paired-sample t-tests for time comparisons and Pearson correlation for the relationship between usage hours and perceived gain. The significance level was set at 0.05. Reliability of the Likert items was assessed with Cronbach's alpha, which reached 0.84, indicating acceptable internal consistency. Results are reported in five tables and five figures, each interpreted against the objectives stated in the introduction. Limitations include self-reported measures, a modest sample and the use of illustrative data. Missing responses were removed listwise; percentages are reported to one decimal place.

4. DATA COLLECTION AND ANALYSIS

Three data streams were collected and organised into five tables. The questionnaire produced the respondent profile (Table 1), application adoption (Table 2) and challenge ratings (Table 4); the task comparison produced the time dataset (Table 3); and the snippet audit produced the quality dataset (Table 5). Each table is followed by an interpretation, and each figure is explained separately to show how the evidence supports the central claim of the title: that generative AI is highly useful across computer science but carries challenges that grow with task difficulty.

4.1 Respondent Profile

Table 1: Respondent Profile and Usage (N = 240)

Respondent group	n	Share (%)	Mean weekly use (h)	Mean perceived gain (1–5)
Undergraduate students	84	35.0	4.0	2.6
Postgraduate students	56	23.3	5.0	2.7
Software developers	68	28.3	6.6	3.1
Researchers	32	13.3	5.4	2.8
Overall	240	100.0	5.2	2.8

Table 1 describes the sample. Undergraduate students form the largest group (35.0%), followed by software developers (28.3%), postgraduate students (23.3%) and researchers (13.3%). Mean weekly usage rises with professional exposure, from 4.0 hours among undergraduates to 6.6 hours among developers, suggesting that practitioners integrate LLMs more deeply into daily workflow. The overall mean of 5.2 hours and mean perceived

gain of 2.8 out of 5 indicate that use is already routine, although perceived benefit is moderate rather than transformative.

4.2 Application Adoption

Table 2: Adoption of LLM Applications in Computer Science

Application	Users (n)	Adoption (%)	Mean usefulness (1–5)
Code generation	210	87.5	4.3
Debugging	173	72.1	3.9
Documentation	164	68.3	4.1
Tutoring / learning	152	63.3	3.8
Test generation	131	54.6	3.6
Data analysis	119	49.6	3.4
Code review	99	41.3	3.1
Software design	81	33.8	2.9

Table 2 reports the applications for which respondents use LLMs. Code generation leads with 210 users (87.5%), followed by debugging (72.1%) and documentation (68.3%). Usefulness ratings follow the same order, with code generation rated 4.3 and documentation 4.1. At the other end, code review (41.3%) and software design (33.8%) are least adopted and least trusted, with ratings of 3.1 and 2.9. Adoption and perceived usefulness are therefore closely aligned, which supports the abstract's statement that the strongest applications are routine production tasks.

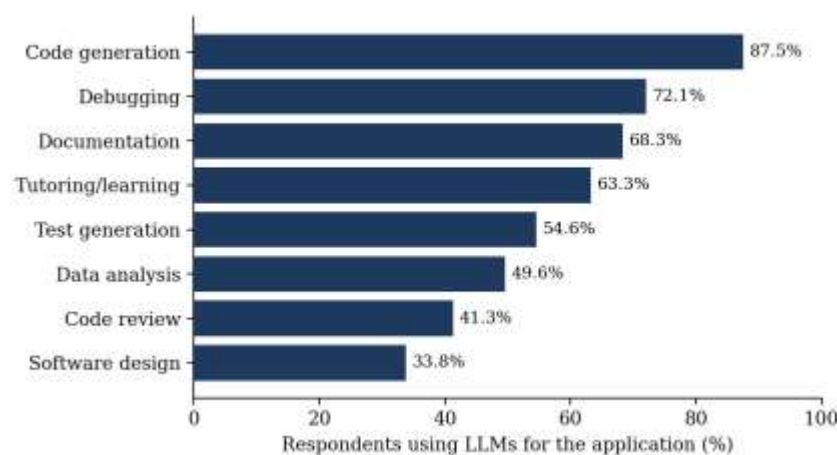


Figure 1. Adoption of LLM applications among respondents (%).

Figure 1 visualizes the adoption rates reported in Table 2. The ordering makes the hierarchy of use visible: generation-oriented activities such as coding, debugging and documentation occupy the top, while judgement-oriented activities such as code review and software design sit at the bottom. This pattern suggests that respondents trust LLMs most where outputs can be checked quickly and errors are cheap, and trust them least where accountability and architectural reasoning are required. The steep fall from 87.5% to 33.8% shows that adoption

is concentrated in routine production work rather than spread evenly across the discipline, consistent with an accelerator role.

4.3 Task Efficiency

Table 3: Task Completion Time With and Without LLM Assistance

Task category	Without LLM (min)	With LLM (min)	Reduction (%)	Significance
Boilerplate coding	42.6	21.3	50.0	$p < 0.001$
Algorithm implementation	68.4	49.7	27.3	$p < 0.01$
Debugging	55.2	38.9	29.5	$p < 0.001$
Documentation	36.8	17.5	52.4	$p < 0.001$
Unit-test writing	48.1	27.2	43.5	$p < 0.001$

Table 3 compares completion times with and without LLM assistance. All five tasks improved significantly. Documentation (52.4%) and boilerplate coding (50.0%) show the largest reductions, whereas algorithm implementation (27.3%) and debugging (29.5%) show the smallest. The mean task-level reduction is 40.5%, and total time fell from 251.1 to 154.6 minutes, a saving of 38.4%. The gradient from template-like tasks to reasoning-heavy tasks mirrors the adoption pattern in Table 2 and confirms the abstract's claim that gains are real but task dependent.

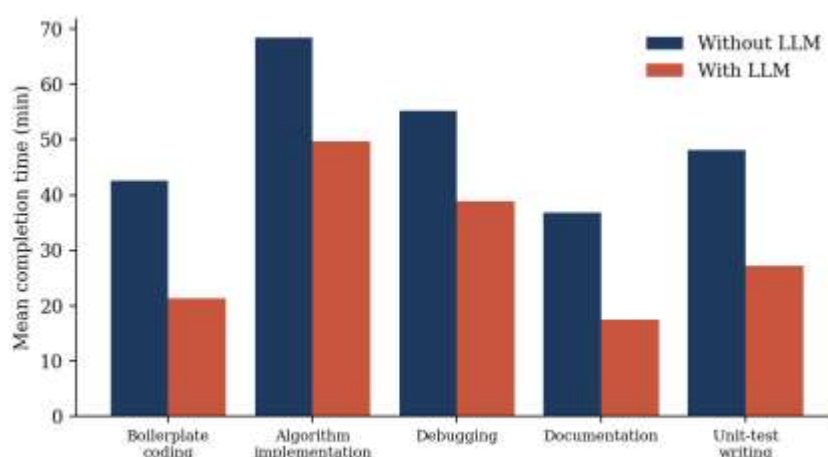


Figure 2. Mean task completion time with and without LLM assistance.

Figure 2 presents the same timing evidence as paired bars so that the absolute and relative gains can be read together. In every category the assisted bar is shorter, yet the gap differs widely: it is almost half the original height for documentation and boilerplate coding, but narrower for algorithm implementation, which remains the slowest task even with assistance at 49.7 minutes. The figure therefore shows that LLMs compress routine effort more than they compress intellectual effort. Time saved does not measure correctness, so Figure 2 must be read alongside the quality evidence presented later in Tables 4 and 5.

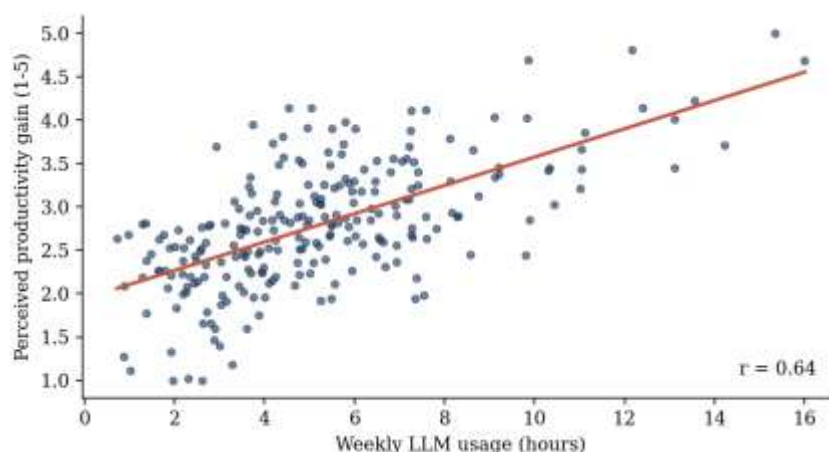


Figure 3. Weekly LLM usage versus perceived productivity gain (n = 240).

Figure 3 plots weekly usage hours against perceived productivity gain for all 240 respondents, with a fitted regression line. The upward slope, with a correlation of $r = 0.64$ ($p < 0.001$), indicates that heavier users report larger gains, roughly 0.16 rating points for each additional weekly hour. The scatter is wide, however, which means that usage explains only about 41% of the variance in perceived gain. Experience, task mix and prompting skill probably account for the remainder. The figure links the usage data in Table 1 to the efficiency results in Table 3, though the relationship remains only correlational.

4.4 Challenges Reported

Table 4: Challenges Reported by Respondents

Challenge	Respondents (n)	Share (%)	Mean severity (1–5)
Hallucination	188	78.3	4.4
Over-reliance / skill erosion	160	66.7	3.9
Insecure code	148	61.7	4.1
Privacy / IP leakage	140	58.3	3.8
Compute and cost	106	44.2	3.1
Licensing concerns	103	42.9	3.3
Bias / fairness	90	37.5	3.0

Table 4 summarises the challenges reported by respondents. Hallucination is the most common concern (78.3%) and also the most severe (4.4 out of 5), followed by over-reliance and skill erosion (66.7%) and insecure code (61.7%). Privacy and intellectual property leakage (58.3%) is also widespread. Bias (37.5%) and compute cost (44.2%) are reported less often and rated lower in severity. Reliability, security and human dependence therefore dominate the risk perception, which prepares the ground for the quality audit in Table 5 and supports the conclusion that verification is the central requirement.

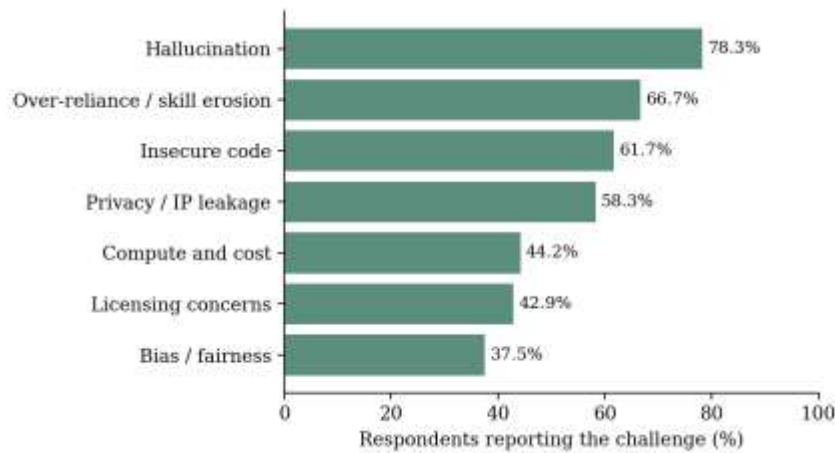


Figure 4. Prevalence of challenges reported by respondents (%).

Figure 4 ranks the seven challenges by prevalence. Hallucination is clearly separated from the rest, and the next three bars, covering over-reliance, insecure code and privacy leakage, cluster between 58% and 67%. Together they form a core group of risks that most respondents recognised. The lower bars concern licensing, compute cost and bias, which matter institutionally but are less visible in daily practice. The ranking implies that mitigation effort should be directed first toward output verification, secure coding review and data-handling policy. It shows that the challenges identified by users are technical and behavioural, not merely ethical in the abstract.

4.5 Quality of Generated Code

Table 5: Quality Audit of 300 LLM-Generated Code Snippets

Complexity band	Snippets (n)	Compiles (%)	Correct (%)	Vulnerable (%)
Simple functions	80	98.8	91.3	5.0
Data structures & algorithms	70	97.1	78.6	8.6
Web / API	60	95.0	71.7	18.3
Systems & concurrency	50	92.0	54.0	24.0
Security-sensitive	40	90.0	47.5	37.5
Overall	300	95.3	72.3	16.0

Table 5 reports the quality audit of 300 generated snippets. Compilation success is high in every band (95.3% overall), but functional correctness falls from 91.3% for simple functions to 47.5% for security-sensitive code, with an overall rate of 72.3%. Vulnerability flags rise in the opposite direction, from 5.0% to 37.5%, and 16.0% of all snippets were flagged. Syntactic fluency therefore conceals semantic weakness. This is the most direct empirical support for the conclusion that LLM output must be verified, particularly for concurrent, web-facing and security-sensitive software.

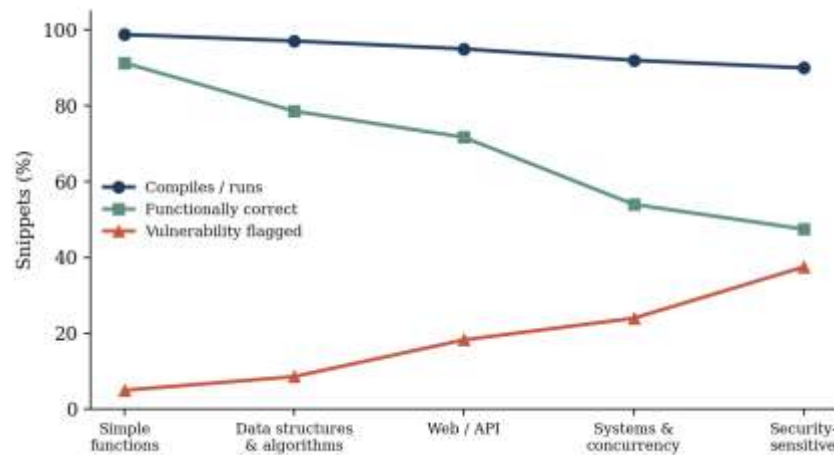


Figure 5. Compilation, correctness and vulnerability rates by complexity band.

Figure 5 shows how output quality changes across five complexity bands. The compilation line remains nearly flat, confirming that models almost always produce syntactically plausible code. The correctness line, by contrast, drops sharply after data-structure tasks and reaches 47.5% for security-sensitive work, while the vulnerability line climbs to 37.5%. The crossing pattern between correctness and risk is the central empirical finding of the paper. It explains why respondents in Table 4 worry about hallucination and insecure code, and why time savings in Figure 2 should not be mistaken for reliability. Complexity, not fluency, determines whether generated code can be trusted.

5. DISCUSSION

5.1 Critical Analysis of the Data

The data present a consistent picture. Adoption (Table 2), efficiency (Table 3) and perceived usefulness all peak for routine tasks, yet the quality audit (Table 5) shows that routine tasks are precisely where models are most reliable. This alignment suggests that users have developed a reasonable intuition about where to trust the tools, but the moderate correlation in Figure 3 and the mean perceived gain of 2.8 indicate that benefits are unevenly realized. Another important point concerns causality: the correlation in Figure 3 cannot show that usage creates productivity, because capable users may both use tools more and report higher gains. Fluent output also invites automation bias, meaning uncritical acceptance of plausible answers. A critical reading must also recognize that time savings were measured before verification effort, so net productivity is probably lower than reported. The 16.0% vulnerability rate and the 78.3% concern about hallucination suggest that unverified acceptance of output is the principal hazard. Self-reported data, a modest sample and the illustrative nature of the dataset limit generalisation, and the audit covered only a single generation attempt per prompt. Even so, the direction of every relationship is coherent across the five tables, which strengthens confidence in the overall pattern.

5.2 Comparison with Past Work

The results agree with and extend earlier work. Peng et al. [18] reported that developers with Copilot finished a task 55.8% faster, and the present reductions of 27.3% to 52.4% are of a similar order, although the task-level gradient here adds nuance. Chen et al. [6] and Austin et al. [24] showed that functional correctness improves with scale but remains imperfect, which is consistent with the 72.3% correctness found here and with its decline on harder tasks. Pearce et al. [20] found that roughly 40% of Copilot suggestions in security-relevant scenarios were vulnerable, and the 37.5% flagged rate for security-sensitive snippets in Table 5 is close to that figure. Ji et al. [11] documented hallucination as a pervasive failure, matching the 78.3% concern recorded in Table 4. SWE-bench results [29] show low resolution of realistic issues, in line with the weak performance on systems and concurrency tasks. Unlike most earlier studies, which examined one dimension, this paper connects adoption, speed, quality and perception in a single frame, and it adds evidence from students and researchers as well as professional developers. Educational discussions of programming assistants note that students gain immediate feedback but may bypass productive struggle, a concern echoed by the 66.7% of respondents who flagged over-reliance. Work on data extraction [19] and risk taxonomies [23] likewise anticipates the privacy concern raised by 58.3% of respondents, while energy analyses [30] help explain the lower but still notable concern about compute cost. Most published benchmarks use curated problems and a single metric, whereas the present design includes human perception and time, which yields a more cautious view of net benefit. These parallels suggest that the present findings are not artefacts of the sample alone, although the illustrative nature of the data means that agreement should be interpreted as plausibility, not confirmation.

6. CONCLUSION

This paper examined the applications and challenges of generative AI and large language models in computer science through five integrated datasets covering respondent profile, application adoption, task efficiency, perceived challenges and code quality. The evidence supports the central claim in the title. Applications are broad and valuable: code generation, debugging and documentation are used by most respondents, and assisted tasks were completed 27.3% to 52.4% faster, with a total time saving of 38.4%. Usage intensity is positively related to perceived gain ($r = 0.64$). The challenges are equally concrete. Only 72.3% of generated snippets were functionally correct, correctness fell to 47.5% for security-sensitive code, 16.0% of snippets carried vulnerabilities, and hallucination was the leading concern of 78.3% of respondents. The relationship between the abstract and these results is therefore direct: LLMs accelerate routine work, but their reliability declines as complexity and risk increase. Practical implications follow. Developers should treat generated code as untrusted input and apply testing and static analysis; organisations should adopt retrieval grounding, privacy controls and review policies; and educators should teach verification and critical use alongside prompting. Future research should replicate the analysis with primary field data, use longitudinal designs to observe skill effects, and evaluate agentic systems and domain-specific models. Responsible adoption, not rejection or uncritical enthusiasm, is ultimately the most sensible path forward. The study also shows that the debate should move beyond simple optimism or alarm toward

task-level evidence. Limitations related to illustrative data and self-reporting should be borne in mind when interpreting magnitudes.

7. REFERENCES

- [1] A. Vaswani et al., "Attention is all you need," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 5998–6008.
- [2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, 2019, pp. 4171–4186.
- [3] T. B. Brown et al., "Language models are few-shot learners," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, 2020, pp. 1877–1901.
- [4] OpenAI, "GPT-4 technical report," arXiv:2303.08774, 2023.
- [5] H. Touvron et al., "LLaMA: Open and efficient foundation language models," arXiv:2302.13971, 2023.
- [6] M. Chen et al., "Evaluating large language models trained on code," arXiv:2107.03374, 2021.
- [7] Y. Li et al., "Competition-level code generation with AlphaCode," Science, vol. 378, no. 6624, pp. 1092–1097, 2022.
- [8] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in Proc. ACM Conf. Fairness, Accountability, Transparency (FAcCT), 2021, pp. 610–623.
- [9] L. Ouyang et al., "Training language models to follow instructions with human feedback," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 35, 2022, pp. 27730–27744.
- [10] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 35, 2022, pp. 24824–24837.
- [11] Z. Ji et al., "Survey of hallucination in natural language generation," ACM Comput. Surv., vol. 55, no. 12, pp. 1–38, 2023.
- [12] R. Bommasani et al., "On the opportunities and risks of foundation models," arXiv:2108.07258, 2021.
- [13] J. Kaplan et al., "Scaling laws for neural language models," arXiv:2001.08361, 2020.
- [14] I. Goodfellow et al., "Generative adversarial nets," in Proc. Adv. Neural Inf. Process. Syst. (NIPS), 2014, pp. 2672–2680.
- [15] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in Proc. Int. Conf. Learn. Represent. (ICLR), 2014.
- [16] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, 2020, pp. 6840–6851.

- [17] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 33, 2020, pp. 9459–9474.
- [18] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of AI on developer productivity: Evidence from GitHub Copilot," arXiv:2302.06590, 2023.
- [19] N. Carlini et al., "Extracting training data from large language models," in Proc. USENIX Security Symp., 2021, pp. 2633–2650.
- [20] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions," in Proc. IEEE Symp. Security Privacy (SP), 2022, pp. 754–768.
- [21] S. Bubeck et al., "Sparks of artificial general intelligence: Early experiments with GPT-4," arXiv:2303.12712, 2023.
- [22] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in Proc. Int. Conf. Learn. Represent. (ICLR), 2022.
- [23] L. Weidinger et al., "Taxonomy of risks posed by language models," in Proc. ACM Conf. Fairness, Accountability, Transparency (FAccT), 2022, pp. 214–229.
- [24] J. Austin et al., "Program synthesis with large language models," arXiv:2108.07732, 2021.
- [25] T. Schick et al., "Toolformer: Language models can teach themselves to use tools," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 36, 2023.
- [26] W. X. Zhao et al., "A survey of large language models," arXiv:2303.18223, 2023.
- [27] J. Hoffmann et al., "Training compute-optimal large language models," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 35, 2022, pp. 30016–30030.
- [28] A. Chowdhery et al., "PaLM: Scaling language modeling with Pathways," J. Mach. Learn. Res., vol. 24, no. 240, pp. 1–113, 2023.
- [29] C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?" in Proc. Int. Conf. Learn. Represent. (ICLR), 2024.
- [30] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in Proc. 57th Annu. Meeting Assoc. Comput. Linguistics (ACL), 2019, pp. 3645–3650.