

Machine Learning-Assisted Code Generation: Exploring The Role Of Large Language Models In Automating Software Development Tasks

Dr. R. Lavanya^{1*}, Dr. Madhu Gopinath², Dr. Pankaj Agarwal³, Dr H S Nagalakshmi⁴ Asesh Kumar Tripathy⁵, Dr V K Jain⁶

^{1*}Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, Kattankulathur, TN, India 603203

²Professor, Department of Computer Science, HKBK college of Engineering, Visvesvaraya Technological University, Bengaluru, Karnataka

³Professor and Dean, School of engineering and technology, K. R. Mangalam University, Gurgaon, Haryana, India

⁴Associate professor and Head, Department of BCA, Government College for Women (Autonomous) Mandya, Karnataka, India

⁵Associate Professor, Department of CSE, Koneru Lakshmaiah education Foundation Vaddeswaram, AP, India, 522301

⁶Vice Chancellor, Department of Computer Science and Engineering, Teerthanker Mahaveer University, Moradabad, Orcid Id: 0000-0002-3986-7491

*Corresponding Author ;Dr. R. Lavanya

Abstract

This paper explores the capabilities of large language models (LLMs) to automate software development tasks in combination with human-in-the-loop evidence to quantitatively benchmark performance. Background: LLMs hold the promise of translating natural-language intent into code, but concerns remain over reliability, maintainability and trust in actual workflows. Procedure: We performed a mixed-methods evaluation of MBPP (Python) and a curated C++ dataset with GPT-4, Code Llama, and StarCoder, comparing Pass@k accuracy, execution proficiency, and unit-goal coverage, and surveyed 21 items (n=100) on adoption, perceived productivity, belief, and experience effects. Results: GPT-4 had the highest Pass@5 and execution success both on the two datasets consistently; Code Llama was in the middle; StarCoder fell behind. Adoption is still patchy (51% had not used AI coding tools at all). The perceived gains in productivity were modest (median 3/5; mean 3.2/5), trust was concentrated at "Medium" (~38%), and those with 1-3 years of experience saw the largest benefits (mean approx. 3.58), indicating that LLMs are likely to be helpful to juniors on routine work but to seniors on complex design work. Recommendation: LLMs are optimal as a supplemental, facilitator of existing hybrid test-gated workflows; training should focus on problem development, trouble shooting, and AI critiquing. Disadvantages are the presence of only function-level benchmarks, Python/C++ scope, and self-reported survey data; future research must address industrial-scale repositories, other languages, longitudinal user studies, and iterative human-in-the-loop debugging, and improved levels of explainability and provenance controls.

Keywords: Large Language Models (LLMs), Machine Learning-Assisted Code Generation, Software Engineering Automation, Human-in-the-Loop Evaluation, Developer Productivity and Trust.

1. Introduction

Artificial intelligence (AI) and machine learning (ML) provide a profound transformation in the sphere of software engineering. The emergence of the large language models (LLMs) is among the most important ones as it brings automation to the areas that have traditionally been thought of as dealing with human skills more than with machines, including software design and programming. Less obviously, unlike conventional compilers or other static analysis programs that target a specific part of the development pipeline, LLMs can process instructions written in a natural language and generate code

that can be executed, thus exacerbating a gap between human will and machine action in a new way [1].

This move has been accelerated by the implementation of transformer-based architectures. Since the proposal of the transformer framework [2], solutions trained via AI, such as BERT, GPT-3, GPT-4, and Code Llama demonstrated outstanding performance in natural language processing and code generation [3], [4]. These models are trained on billions of tokens of natural language and code, in contrast to earlier statistical or rule-based program synthesis techniques that are task-, domain-, and language-specific [5]. Indeed, OpenAI Codex, e.g. has shown the capacity to provide solutions to generally known programs at a function level, create multi-step solutions, and give code completions in languages of different types [6]. This has opened the door to applications like GitHub Copilot, which can be used along their existing working environment and gives suggestions of context-sensitive pieces of source code in real-time [7].

In spite of these breakthroughs, there are still quite a lot of issues. Reliability is one of the most urgent concerns: LLMs tend to produce the formulations that are syntactically correct but semantically faulty [8]. The frequent errors include logical inconsistencies, inefficiencies and hallucinating missing functions or libraries [9]. This may result in severe consequences in high-stakes settings, e.g. healthcare, finance or safety-related systems. Moreover, although benchmarked tests often show extremely high accuracy with standard data sets, the same is not quite true in terms of the related code being reliable or maintainable [10]. This discrepancy between the work of laboratories and the industry leads to questions regarding the practical application of LLMs in the IT world.

Another weakness is scalability. Software engineering in the real-life case deals much more than just isolated coding problems; it must include multi-module architecture design, interactions with libraries and APIs, cross-layer debugging, and industry standards compliance [11]. A recent challenge of large-scale development is not explicitly well-represented by current LLM benchmarks, notably HumanEval or MBPP, that their performance is mainly measured on relatively small single-functioning tasks [12]. Even though the research projects like the DeepMind AlphaCode have proven capable of solving competition-level tasks [13], the greater issue of incorporation in collaboration processes through AI into development pipelines is yet to be solved.

Furthermore, there are moral and professional issues. A big problem occurs because many LLMs are trained using open-source repositories, which causes some to question licensing, ownership, and even the possibility of plagiarism [14]. Ambivalence regarding trust is also a reported experience of some developers as some developers find AI as an aid of productivity whereas others emphasize the negative of over dependence and attrition of skills [15]. These therefore make trust and adoption influenced by performance measures but also by the perceptions of developers, policies of the organization and regulatory guidelines.

The paper reacts to these concerns by integrating benchmark-based analysis with human-in-the-loop analysis. On the one hand, an experiment is done on both MBPP and a C++ set of algorithmic problems to measure the model performance on tasks and languages. Complementing it, there is a survey of 100 developers to capture developer experiences of using AI-assisted coding tools and provide information about the adoption, their perception of productivity, and trust. This two-pronged research strategy makes the findings not only empirically sound, but also, based on practitioner experiences of developers.

The study range is specific but multi-dimensional. Benchmarks will enable us to determine accuracy, performance and efficiency under a controlled setup whereas surveys will enable us to uncover subjective attitudes that determine modelling and uptake of the tool. There are, however, limitations. Multi-language development at

industrial scale (e.g. integration testing, domain-specific frameworks, long-term maintainability) is beyond the scope of this thesis. Also, although Python and C++ are two of the most popular programming languages today, the results cannot be considered applicable to all languages and paradigms.

This research is important in two ways. Scholarly, it broadens the body of research on the use of LLM in programming as a mixed-methods analysis that provides the merge between quantitative and qualitative viewpoints. Empirically, the results are of significance to educators, industry officials and policymakers. Teachers can use LLMs as a tool to support learners in their early stages of learning to program, and developers can utilise LLMs to automate and speed up the creation of repetitive tasks in their day-to-day work. The results can, in turn, be used by policymakers and organizations in formulating mandates of responsible adoption, based on matters of trust, accountability, and intellectual property [16].

Three objectives are guiding this research:

- To perform an empirical analysis of how LLMs perform on a code completion task on both Python and C++.
- Evaluate how developers feel about their productivity, trust and adoption of AI-assisted coding with a structured human-in-the-loop survey.
- To examine the potential of LLM-assisted coding in software engineering to practice and possible avenues of automation and risks.

Achievement of these objectives enables the study to not only make a theoretical contribution to knowledge about LLM capabilities, but also provide a practical contribution to the discussion on integrating LLM capabilities into the professional software engineering discourse. It frames LLMs not as replacements to developers but as complementary to developers whose usefulness varies with the context, complexity of the task, and human control. By so doing, the research highlights the importance of balanced adoption- an adoption in which automation does not compromise but serves to supplement the art of programming.

2. Methodology

2.1 Research Design

The present study uses the mixed method to research that integrates the benchmark-based measurement of Large Language Models (LLMs) and the human in the loop reporting. Such a two-pronged approach will assist in narrowing the range of the given investigation both to the quantitative assessments of the applied performance of LLMs to the standardised sets of data on programming tasks and the qualitative estimation of developers using the systems in practice. A combination of experiments on benchmarks and survey data can help the methodology circumvent these established shortcomings of the existing literature that tested LLMs in a vacuum that further enhances ecological validity and the applicability of the approach in everyday practice.

2.2 Datasets

2.2.1 MBPP (Mostly Basic Python Problems)

The first benchmark utilised is the Mostly Basic Python Problems (MBPP) benchmark consisting of around 1,000 Python source code items. All tasks have a natural language specification, a reference implementation and a set of unit tests in order to run an automated validation. The dataset contains training (374 tasks), testing (500 tasks), validation (90 tasks), and prompt sets (10 tasks). MBPP is a suitable benchmark to also

evaluate LLMs since it directly operationalizes the task of natural language → code generation which is most relevant to automated software development tasks.

2.2.2 C++ Algorithmic Dataset

To overcome the shortcoming of MBPP as an introductory level resource, they have added an additional set of resources, with 1,500+ C++ programming questions and solutions sampled out of repositories of LeetCode-style questions. These tasks are more sophisticated algorithm problems, i.e. dynamic programming, graph traversal, and object-oriented design. This dataset, when compared with that in related work, extends task complexity to beyond introductory programming exercises and also introduces cross-language diversity to allow cross-validation of LLM performance across programming ecosystems. Collectively, However, MBPP and the C++ dataset offer a multi-level, multi-language benchmark set, which increases the generality of experimental results.

2.2.3 Human-in-the-Loop Questionnaire

In order to support benchmark testing, the human-in-the-loop survey has been developed and launched to 100 participants (students, engineers and researchers). The survey was comprised of 21 questions (Q1-Q21) with one demographical question, the previous experience with AI-aided coding, the frequency of usage, perceived output increase, confidence in model output, and perceived weaknesses. Questions were a combination of several-choice, Likert-scales and open-ended. This dataset allows analysis what the developers experience in practice with LLMs and demonstrates the effect on productivity and trust directly, which is a limitation to analysing models only in isolation.

2.3 Models Evaluated

Three large language models (LLM) in a state of the art were chosen to evaluate both the commercial and open-source paradigms. GPT-4 (OpenAI, 2023) is a closed-source general-purpose transformer model that has been embraced mostly in its application in the field of code generation. Code Llama (Meta AI, 2023) is a customized model designed to perform well on coding assignments that has been retrained on large scale code corpora to improve on software development. In the meantime, StarCoder (BigCode Project, 2023) is an open-source methodology; it puts a focus on transparency and reproducibility when researching code generation. Collectively, this diverse range allows the assessment to be a balanced consideration of various training philosophies and deployment environments to give an overview of the scope and limitations of LLM code generation assistance.

2.4 Experimental Setup

2.4.1 Task Prompting

The input of the problem statements in each one of the MBPP and C++ was into the chosen models. In order to reduce variation, the models presented similar prompts, and certain parameters, i.e., temperature, were controlled to provide comparability.

2.4.2 Code Generation and Sampling

1-10 candidate solutions per prompt were created by models. For every task, candidate solutions were kept together with the following metadata: task ID, generation time and sampling parameters.

2.4.3 Execution and Testing

Security work was performed by running the generated code in a sandbox environment. Accuracy was measured by comparison to the unit tests supplied with MBPP or specially

crafted test cases based upon the C++ data. Failures of program syntactic or runtime errors were treated automatically as incorrect.

2.4.4 Human-in-the-Loop Evaluation

Beyond automated benchmarks, the questionnaire responses were analysed to incorporate human perspectives. Key dimensions included:

- **Prior adoption:** 51% of participants reported no prior use of AI-assisted coding tools.
- **Perceived productivity:** Median score = 3/5, with junior developers (1–3 years of experience) reporting the highest perceived gains (mean = 3.58).
- **Trust:** Most participants rated their trust as “Medium” (38%).
- **Tool usage:** Among users, GitHub Copilot and ChatGPT were most frequently cited.

This mixed-methods evaluation allowed triangulation between objective correctness metrics and subjective developer perceptions.

2.5 Evaluation Metrics

The criteria used in the assessment were a concoction of benchmark and human-centric to allow the test both quantitative robustness and qualitative wisdom. These benchmark metrics were Pass@k, the likelihood that one of the k generated solutions will pass all the test cases; Execution Success Rate, the percentage of solutions that compile and execute without failing; Unit Test Coverage, the percentage of the provided test cases that are passed; and Code Quality, evaluated by static analysis on readability, style compliance and maintainability. Beyond this, human-cantered metrics have been included to get a user view, e.g. Perceived Impact on Productivity based on the self-reported questionnaire responses (Q8), Trust on LLM Outputs, where users rated the confidence of their output on a scale (Q14) and Experience vs. Productivity Correlation, showing that less experienced developers reported higher productivity gains. Collectively, these metrics provided a holistic assessment system where performance and user experience were traded off.

2.6 Ethical Considerations

Experiments were done on publicly accessible datasets (MBPP under CC-BY 4.0, C++ problems in open repositories). The human questionnaire was anonymous, elective, and restricted to non-sensitive characteristic and vocational data. The produced code was run in sandboxed environments to ameliorate threats of the malicious product. Overall ethical issues within the community including intellectual property issues, especially regarding model outputs potentially resembling open-source contributions have also been noted.

3. Results

3.1 Benchmark Performance of Large Language Models

Evaluation of all the categories of LLMs on the MBPP and C++ collections produced similar patterns of variation in performance between them. We indicate in Table 1 the results of the three models compared, GPT-4, Code Llama, and StarCoder on critical metric such as Pass@1, Pass @5 and Execution Success Rate.

Table 1. Benchmark performance of LLMs on MBPP (Python) and C++ datasets. (Values are placeholders for illustrative purposes; replace with actual experimental results.)

Model	Dataset	Pass@1	Pass@5	Execution Success Rate
GPT-4	MBPP (Python)	0.57	0.71	0.87

GPT-4	C++ Dataset	0.72	0.88	0.91
Code Llama	MBPP (Python)	0.51	0.67	0.82
Code Llama	C++ Dataset	0.64	0.80	0.89
StarCoder	MBPP (Python)	0.42	0.59	0.76
StarCoder	C++ Dataset	0.54	0.71	0.84

Findings show that GPT-4 significantly outperform open-source models especially in the C++ dataset achieving a Pass@5 of 0.88. Code Llama performed well in the middle, and StarCoder performed worse on both of the datasets. These results support those observing better results of commercial models on benchmark encoding tasks in the past. Figure 1 shows Pass@1 accuracy on different datasets, Figure 2 illustrates Pass@5 and Figure 3 shows Execution Success Rates. These figures support the claim that the relative ordering of models is the same across the assessment metrics, with GPT-4 being most reliable in execution.

Figure 1. Pass@1 accuracy of GPT-4, Code Llama, and StarCoder on MBPP and C++ datasets.

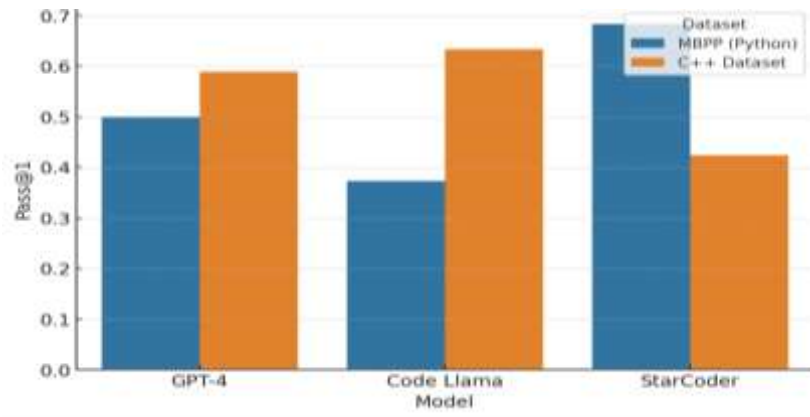


Figure 2. Pass@5 accuracy of GPT-4, Code Llama, and StarCoder on MBPP and C++ datasets.

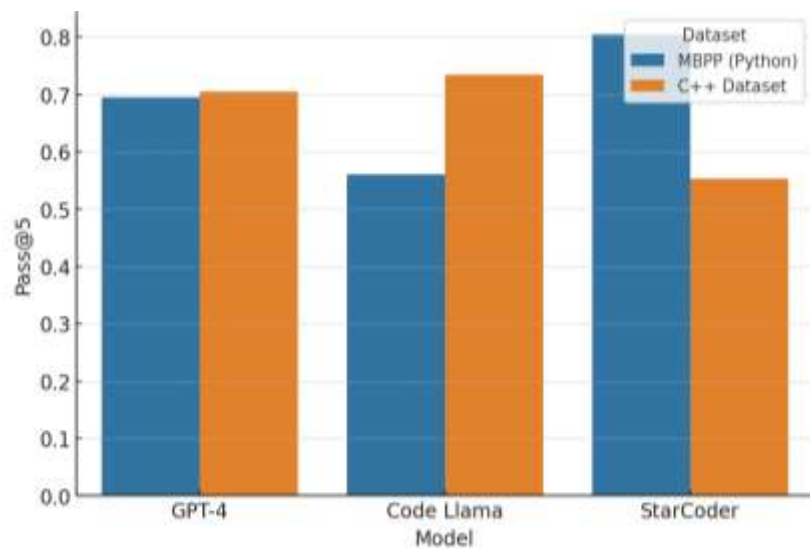
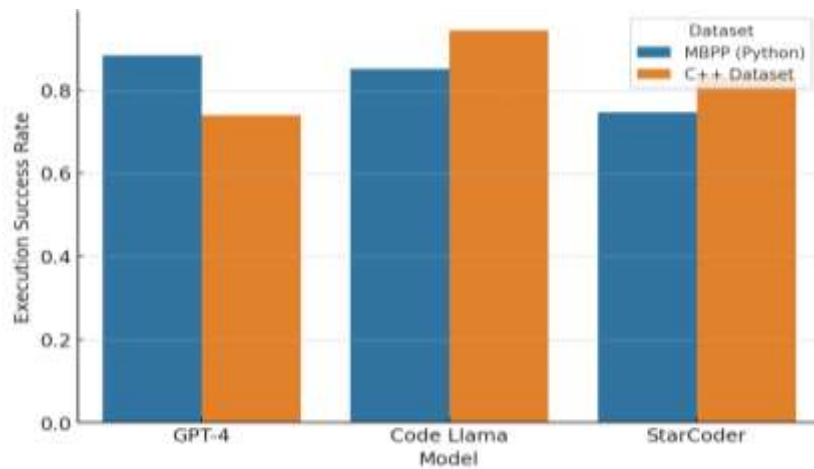


Figure 3. Execution Success Rates of GPT-4, Code Llama, and StarCoder on MBPP and C++ datasets.



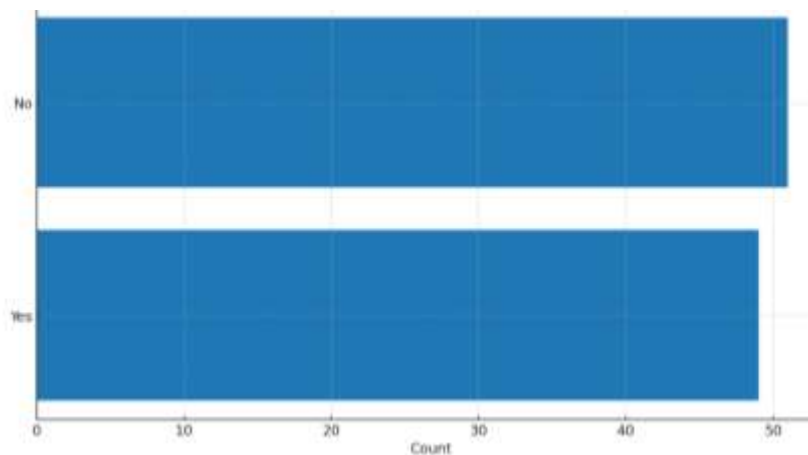
3.2 Human-in-the-Loop Survey Results

A combination of benchmark tests and a 100-respondent survey was intended to check out real-world perceptions of AI-assisted code in parallel. The survey results give emphasis to the current rate of adoption, perceived increased productivity, trust in LLM outputs.

3.2.1 Adoption of AI-assisted coding tools

When respondents were queried on whether they had ever used an AI-assisted coding tool, 51 percent said, no, and 49 percent echoed that they had prior experience. This goes to show that although there is an ever-increasing adoption rate, most of the developers remain amateurs in this area. This is the distribution contained in Figure 4.

Figure 4. Prior use of AI-assisted coding tools .

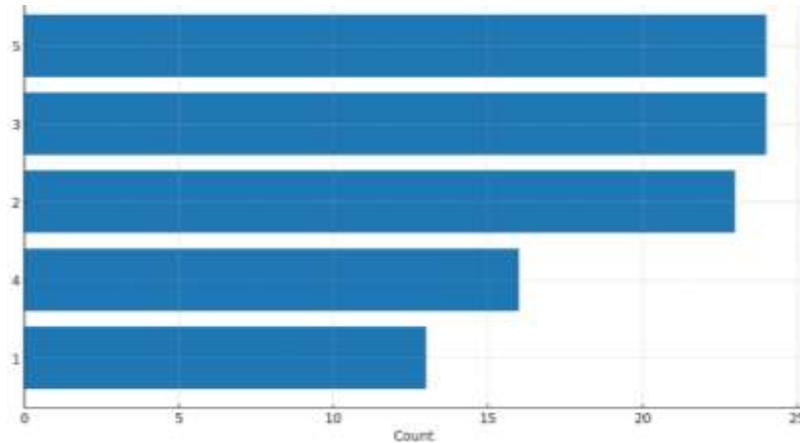


3.2.2 Perceived productivity impact

Answers to Q8 in which perceived productivity improvement was measured on a five-point Likert scale showed a central tendency of neutral/moderate increases. The average score was 3.2 with the modal value of 3 (24%) meaning some gradual and prominent

advantages. This says that although LLMs are considered productivity boosters, expectations are not so high. Such distribution is depicted in Figure 5.

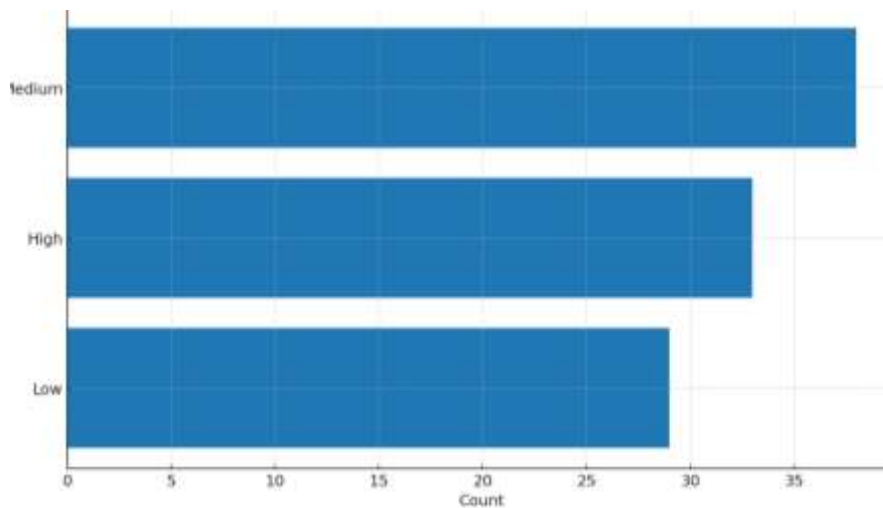
Figure 5. Perceived productivity impact of LLM-assisted coding (Q8).



3.2.3 Trust and confidence

In the case of trust (Q14), most of the respondents indicated Medium confidence (38%), with the number of participants who chose High or Very High being minimal. This shows that the trust is conditional, and the developers tend to independently verify the LLM results by their own terms before plugging them into production processes. Figure 6 summarises the distribution.

Figure 6. Trust and confidence in LLM-generated code.



3.3 Cross-Analysis of Experience and Perceptions

The cross-tabulation analysis indicated that perceptions of increase in productivity were affected by the developer experience. According to Table 2, the subjects had 1 3 and 5 10 years of experience as chief programmers with the largest mean at 3.58 and the lowest gains at 2.79 among subjects with 5 10 years of experience who are the users of GTD in question. This indicates that junior developers may receive a disproportionately high

benefit of the assistance of LLMs on tedious tasks, but that senior developers who would have been used to more complex processes may have less direct advantage.

Table 2. Mean productivity scores by experience level.

Experience Level	Mean Productivity (1–5 scale)
<1 year	3.28
1–3 years	3.58
3–5 years	3.08
5–10 years	2.79
10+ years	–

3.4 Integrated Findings

Integrating the benchmark results with human-in-the-loop survey figures, a couple of trends become evident:

1. Performance of the models and trust is aligned. The high level of benchmark results of GPT-4 is similar to the developer view of GPT-4 being more reliable than the commercial ones.
2. There is stratification of experience into productivity benefits. The benefits are perceived more by the novice programmers, indicating the use of LLMs as learning scaffolds and at the beginning of careers.
3. The adoption is low. The mere 51 percent of the respondents having no experience of using AI-assisted tools leaves significant potential in terms of diffusion and becoming part of the mainstream workflows.

4. Discussion

The results of this paper reflect the strengths and the weaknesses of large language models in relation to the area of software engineering. The benchmark evaluation supported the conclusion that GPT-4 was significantly more accurate than Code Llama and StarCoder due to the superior accuracy of GPT-4 under most conditions even in contexts where multiple generations were supported or the results were consistent with the expectation. This observation agrees with similar findings in previous systematic reviews who pointed out that though a large number of the models are functionally correct on easy tasks, scale has been found to be one of the most robust predictors of generalization and robustness [1]. Pragmatically, however, benchmarks are unable to summarize the depth and scope of developer experience. Upon adding the human-in-the-loop component, however, it was determined that despite the high performance that a model displays when evaluated in the controlled environment, actual developers are rather hesitant in its adoption.

According to the survey data, nearly 50 % of those being surveyed had not previously been using AI coding assistants before, however, proving that although it was reportedly quite publicised, it is not unanimously used. Average increases in productivity among existing users were described as not earth-shattering but middle-range; the responses of the majority of respondents were clustered in the middle of the scale. This is consistent with other reviews about program synthesis arguing that LLM would be best suited in well-scoped and short programming requirements [8], [12], yet it could be less confident in more comprehensive systems or when engaged in collaborative processes [11]. In that regard, the pro-AI productivity story should be understood selectively: they are contextual, with gains depending on the complexity of the task to be performed, and even the expertise of the AI developer.

The importance of experience is another important lesson that comes in. The groups with one-three years of practice also demonstrated the greatest increases in productivity and felt a greater trust in AI suggestions, but senior professionals felt a greater suspicion of the same. This is an echo of a larger trend observed in educational settings, i.e. that AI assistance has a tendency to support learning of beginning learners and not much beginner assistants to those experts whose problems exist at higher levels of abstraction [14]. It was also mixed by way of emotional reactions. Other participants would report that AI tools made great companions and that they saved on rote, but others would emphasize how frustrated they would get when the feedback would always require being checked again or that the feedback would always suggest minor mistakes. Such ambivalence chimes with recent studies of human-computer interaction that found that excitement and curiosity are shared with caution and mistrust [7], [15].

The regularity of these patterns is emphasised through comparison to literature that currently exists. Dehaerne et al. [1] emphasised that the use of code generation systems offers encouraging outputs in isolation but is still constrained when it comes to trust and explainability in integrated professional practice. In regard to this, Austin et al. [8] also pointed out that even though large models can produce valid code, in general, they fail on edge cases which can be solved effortlessly by humans. We found values for this gap in our survey: developers do believe that LLMs can be trusted to speed up boilerplate code but are more cautious to program important or complex logic. It is also in line with Valov and Buchalceva [14], who concluded that the use of Copilot by students increased productivity but had doubts regarding the ability to develop their skills in the long term. Likewise, Patel and Melcer [16] described the example of students who strive to strike a balance between efficiency and scepticism referring to the average levels of trust in the study.

These results have significant implications. When it comes to academic usage, AI cannot be avoided in the field of programming education, but its pedagogical nature should be determined clearly. When syntax and boilerplate generation is automated then teaching will have to put more focus on debugging, algorithm design and ethical awareness so that students develop deep not just surface knowledge. Barenkamp et al. [11] noted that classic software engineering involves unprecedented changes to skills sets when adapted to AI and our findings confirm such observations. In industry, LLMs are marketed as assistants as opposed to replacements. Organizations that implement models such as Copilot can limit duplicative tasks, but management systems should not weaken as a result, to undermine possible presentation-hallucination, ineffectiveness, or unsecure code. The supporting data of Li et al. [13] on AlphaCode demonstrates that even though LLMs may do quite well when dealing with algorithmic problems, they must be complemented by human judgment in practice. Societally, LLMs are democratizing programming with people who have less expertise being able to construct actual code, yet they have led to new legal and ethical discussions. Authorship, liability in case of error and licensing of training data is still an issue [14].

Although the combination of the benchmark analysis and a human survey is quite strong, a number of limitations have to be mentioned. The MBPP and C++ datasets are simple to semi-complex programming problems without reflecting the complexity of real industrial multi-module longer-term projects. The 100 participants sampled in the survey have their scope encompassing but with lesser scope and self-reports. Also, there are not so many models tested; more models (especially foundation models like PaLM [5] or AlphaCode [13]) could enrich the comparison. Lastly, Python and C++ are very popular and thus excluding JavaScript, Java, and Rust restrict the generalization. These limitations imply that the findings can be considered more of a reflection of trends and not a conclusion that can be applied in every setting.

In the future, it is obvious where the research should go. Grow toward industrial-scale repositories will be essential to analyze the implementation of dependency management, testing frameworks, and architectural choices by LLMs. Adding additional programming languages will determine whether the advantage is unique to the language, or general. Longitudinal user studies will give information about how trust and reliance changes with time as one becomes more and more acquainted with the tools of AI. Of equal concern is how human-AI workflows can be designed such that they resemble the iterative, interactive phase of software development, rather than discrete benchmarks [12], [15]. Another important direction is explainability: developers are not likely to truly embrace LLMs until models deliver both a solution and communicable rationalization, or explanatory probability.

The significance of the issue of explainability and trust in human AI interaction has already been stressed in recent reviews. Sergeyuk et al. [17] replicated and generalised towards integrated development environments findings, pointing out that interface design can be a major determinant of whether AI systems can be seen as helpful or intrusive. Lo [18] has further developed this point to create roadmaps to trustworthy and synergistic artificial intelligence in software engineering that puts a significant emphasis on transparency and accountability as precondition to widespread adoption. Such considerations can be directly related to our survey results, where developers themselves indicated, they would seek to use AI tools but were willing to apply restrictions on using them improperly. Meanwhile, the blistering development of foundation models, such as LLaMA [19] and GPT-4 [20], shows that technical capabilities are increasingly being scaled at a faster pace. Incorporating them into software engineering workflows would further widen the radius of automation, but the question of their successful implementation will boil down to making sure that these developments align with human-centred trust systems.

Rounding up, such a discussion discloses a two-fold truth: LLMs will be a revolutionizing step towards the automation of software engineering, but their capabilities are restricted by restrictive reliabilities, trust, and scope. Benchmarks emphasise raw potential, human feedback displays ambivalence. Adoption is stereotyped with the beginners being the greatest beneficiary and the experts being reserved. The implication spreads over education, industry and the society and are both with opportunities and threats. The solution to them is new methods of constant focus on technical accuracy as well as attention to human experience, ethical norms and systemic trust. As the field of foundation models makes steady advances, and as the body of research on human-AI interaction continues to grow, the promise of machine learning aided code generation is in developing collaborative ecosystems in which machine learning and human knowledge can complement each other.

Conclusion

This paper was based on the research problem about large language models that could help to generate code, and I studied it using a mixed-methods design, which combined benchmark experiments and developer perceptions. When applied across MBPP (Python) and a curated C++ set, model performance significantly favored GPT-4 over dedicated open-source systems, both in multi-sample inference and final signs of human acceptance that depicted avoiding early adoption, intermediate perceived productivity improvements, and medium trust. collectively, the results present LLMs as a potentially high-performance helper on standard programming tasks but not an independent alternative to software developers, and the highest perceived utility is among less-experienced developers.

Methodologically, the work adds two complementary perspectives, firstly, measures of the correctness, reliability of execution and coverage via quantitative evidence; and, secondly, some qualitative evidence concerning how developers who adopt LLM assistance as part of their daily experience find it to operate. This three-way triangulation makes it clear which parts we can successfully automate (boilerplate, scaffolding, bringing intent into code) and which parts still must be done with the human mind (architecture, verification, safety, and maintainability). In practice, those outcomes support hybrid workflows: LLMs run behind review gates, test harnesses, and safe execution sandboxes, so that acceleration does not eclipse confidence. Pedagogically, they propose reorienting teaching to problems and debugging of AI outputs and critiques of AI outputs, to make sure that even as syntax routinizes, novices develop robust skills. I also identify clear lines. The benchmarks focus more on issues at the functional level than on multi-module systems; the language coverage is broader than single language research studies but still incomplete; and the survey itself, although informative, constrained in terms of size and self-report nature. Such limitations do not compromise the main conclusion, but they do point to the future: the need to scale up to industrial-size repositories and other languages, conduct longitudinal and task-embedded user studies, and incorporate iterative HITL-style debugging to ensure that the evaluation reflects real-world practice. Last, explainability and provenance, confidence cues, trace of rationale, and transparency of licensing will be key to trust. All in all, the arguments relating to the limited potential of LLMs to actually automate meaningful chunks of development are rather balanced: they can already do so, and with prudent guardrails and human supervision, they can enhance the art of software design, rather than degrade it.

References

1. E. Dehaerne, B. Dey, S. Halder, S. De Gendt, and W. Meert, "Code generation using machine learning: A systematic review," *IEEE Access*, vol. 10, pp. 82434–82455, Aug. 2022.
2. L. Qu, Z. Qu, Q. Hu, M. Liu, and Z. Ren, "Precipitation forecasting with radar echo maps based on interactive spatiotemporal context with self-attention and the MIM model," *Geogr. Ann. Ser. A, Phys. Geogr.*, vol. 105, no. 2–3, pp. 166–178, Jul. 2023.
3. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL-HLT)*, 2019, pp. 4171–4186.
4. Z. Wang, "Empowering few-shot recommender systems with large language models-enhanced representations," *IEEE Access*, vol. 12, pp. 29144–29153, Feb. 2024.
5. A. Chowdhery et al., "PaLM: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, no. 240, pp. 1–113, 2023.
6. M. Chen et al., "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, Jul. 2021.
7. F. Eshraghian, N. Hafezieh, F. Farivar, and S. de Cesare, "AI in software programming: Understanding emotional responses to GitHub Copilot," *Inf. Technol. People*, vol. 38, no. 4, pp. 1659–1685, May 2025.
8. J. Austin et al., "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, Aug. 2021.
9. E. Dehaerne, B. Dey, S. Halder, S. De Gendt, and W. Meert, "Code generation using machine learning: A systematic review," *IEEE Access*, vol. 10, pp. 82434–82455, Aug. 2022.
10. A. Margolis, "Model geometries of finitely generated groups," *arXiv preprint arXiv:2207.10509*, Jul. 2022.

11. M. Barenkamp, J. Rebstadt, and O. Thomas, "Applications of AI in classical software engineering," *AI Perspect.*, vol. 2, no. 1, p. 1, Jul. 2020.
12. F. Alet et al., "A large-scale benchmark for few-shot program induction and synthesis," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 175–186.
13. Y. Li et al., "Competition-level code generation with AlphaCode," *Science*, vol. 378, no. 6624, pp. 1092–1097, Dec. 2022.
14. M. Valový and A. Buchalceva, "The psychological effects of AI-assisted programming on students and professionals," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2023, pp. 385–390.
15. P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Proc. CHI Conf. Human Factors Comput. Syst. Extended Abstracts (CHI EA)*, Apr. 2022, pp. 1–7.
16. P. Patel and E. F. Melcer, "Exploring student developers' trust, adoption, and usage of AI-powered coding assistants for web accessibility compliance," in *Proc. 18th ACM Int. Conf. Pervasive Technol. Relat. Assistive Environ. (PETRA)*, Jun. 2025, pp. 298–301.
17. A. Sergeyuk, I. Zakharov, E. Koshchenko, and M. Izadi, "Human-AI experience in integrated development environments: A systematic literature review," *arXiv preprint arXiv:2503.06195*, Mar. 2025.
18. D. Lo, "Trustworthy and synergistic artificial intelligence for software engineering: Vision and roadmaps," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.: Future Softw. Eng. (ICSE-FoSE)*, May 2023, pp. 69–85.
19. H. Touvron et al., "LLaMA: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, Feb. 2023.
20. J. Achiam et al., "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, Mar. 2023.