iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VIII **Month of publication:** August 2026

DOI: https://doi.org/10.22214/ijraset.2026.84657

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VIII Aug 2026- Available at www.ijraset.com

The Role of AI in Application Development: A Comparative Study with Manual Coding

Perseus Bhavnagri
Student, Wilson College, University of Mumbai, Mumbai, India

Abstract: ***Application development has traditionally depended on manual coding, in which developers write every line of code themselves. This approach offers precision and control, but it is time-consuming, labour-intensive and prone to human error. The emergence of Artificial Intelligence (AI) has introduced tools that generate, test, debug and optimise code, raising the question of how AI-assisted development actually compares with manual practice. This paper presents a comparative study of the two approaches using secondary data from the Stack Overflow Annual Developer Survey 2024, comprising 65,437 responses from developers across 185 countries. Seven hypotheses were formulated covering productivity, job satisfaction, accuracy, compensation, challenges, sentiment and instrument reliability, and were tested using non-parametric methods (Mann-Whitney U, chi-square) at a 5% significance level. The job-satisfaction scale demonstrated excellent internal consistency (Cronbach's alpha = 0.931, 9 items, n = 29,095). The analysis found that 57.6% of respondents currently use AI tools, that 81.0% of adopters identify increased productivity as a benefit, and that 72.0% hold a favourable or very favourable view of AI. However, two widely assumed advantages did not survive testing. The difference in job satisfaction between AI users and manual coders was statistically significant but negligible in magnitude (means 6.97 vs 6.89; Cohen's d = 0.039). The apparent compensation advantage reversed direction once national context was controlled: pooled data showed manual coders earning more, yet within the United States alone the difference disappeared entirely (p = 0.203), indicating that the pooled gap is a confound arising from higher AI adoption in lower-income economies rather than an effect of AI itself. Trust remains the principal barrier, with 65.1% of respondents distrusting AI output and 61.9% reporting that AI tools lack context of their codebase. The study concludes that AI meaningfully augments developer productivity but does not yet demonstrably improve satisfaction or earnings, and that a hybrid human-AI model, supported by governance and training, remains the most defensible direction for application development.***

Keywords: Artificial Intelligence (AI), Manual Coding, AI-assisted Development, Developer Productivity, Software Development Lifecycle, Software Engineering, Hybrid Development Model.

I. INTRODUCTION

A. Background and Context

Application development has always been central to modern technology. From the era of assembly language to today's high-level frameworks, the creation of software has progressed enormously. For most of that history, manual coding, in which developers write each line themselves, was the only option available. This method gives direct control over program behaviour and ensures that developers fully understand the logic and structure of what they build. As applications have grown more complex and delivery timelines have shortened, however, the limits of a purely manual approach have become increasingly visible.

Manual development requires long hours of writing, reviewing and debugging. Human error is common, and correcting it can consume more time than the original implementation. In industries where time-to-market is decisive, such delays translate directly into lost opportunity. The parallel demand for secure, scalable and efficient systems has placed further pressure on development teams. It is in this context that Artificial Intelligence has emerged as a practical response.

B. The Rise of AI-Assisted Development

AI is no longer a speculative technology. It has established itself in healthcare, finance, education and transportation, and in software engineering it now participates directly in the development process. Tools such as GitHub Copilot, ChatGPT-based assistants and AI-driven testing frameworks help developers produce code more quickly and with fewer routine errors. Odeh, Odeh and Mohammed [2] survey the principal techniques underpinning automated code generation, spanning deep learning, evolutionary algorithms and natural language processing, and assess them against accuracy, efficiency, scalability and generalisation. Banh, Holldack and Strobel [1] document how generative AI reshapes software engineering workflows by reducing manual workload and altering the distribution of developer effort.

This shift represents a genuine change in how applications are built and maintained. Rather than relying solely on human reasoning, developers increasingly work alongside machine intelligence.

C. Continuing Relevance of Manual Coding

The arrival of AI does not render manual coding obsolete. Control over architecture, security decisions and business logic still rests with human developers, and AI-generated output requires review before it can be trusted. Cotroneo et al. [3] demonstrate that assessing the correctness of AI-generated code is itself a difficult open problem, proposing an automated method based on symbolic execution to evaluate whether generated code behaves as a reference implementation. Kharma et al. [4] examine security and quality across multiple languages and models, finding that vulnerabilities in AI-generated code remain a material concern. Manual oversight therefore remains necessary rather than optional.

D. Convergence and the Hybrid Model

What emerges from the literature is not replacement but convergence. Terragni, Vella, Roop and Blincoe [5] offer a forward-looking account of AI-driven software engineering in which intelligent automation is embedded across the lifecycle while human judgement governs design and verification. Jackson et al. [6] study creativity and human-AI collaboration, arguing that partnership between developers and generative tools is where productivity gains actually arise. Understanding this shift requires examining both the benefits and the risks of integrating AI into development workflows, measured against traditional practice. That comparison is the purpose of this study.

II. PROBLEM STATEMENT AND NEED FOR THE STUDY

A. Problem Statement

Software applications have become essential to business and daily life, and demand for efficient, secure and scalable systems continues to grow. Traditionally, development has proceeded through manual coding, which ensures control and flexibility but is slow, labour-intensive and error-prone. As complexity increases, manual practice struggles to keep pace with deadlines and expectations.

The rise of AI has introduced tools capable of automating parts of this process, including code generation, testing, debugging and optimisation. Their adoption raises questions that remain inadequately answered:

Can AI-assisted development match manual coding without compromising quality and security?

How reliable and efficient is AI-generated code relative to hand-written code?

What limitations and risks accompany dependence on AI in the development lifecycle?

Much of the existing discussion is either vendor-driven or based on small-scale experiments. There is comparatively little analysis grounded in large-scale, independently collected evidence about what practising developers actually experience. This gap motivates the present study.

B. Need for the Study

The need for this study arises from the speed of AI adoption relative to the evidence base supporting it. Specific justifications are as follows.

Industry relevance. AI tools are being adopted in production environments, yet their long-term effect on code quality, maintainability and security remains uncertain. Evidence is required to determine whether AI complements or displaces human developers.

Educational importance. Students and early-career developers increasingly rely on AI assistants. Understanding the benefits and drawbacks allows educators to design curricula that balance automation against reasoning skill.

Efficiency and innovation. Comparing AI-assisted and manual development clarifies where genuine productivity gains occur and where claimed gains do not withstand scrutiny.

Ethical and reliability concerns. As AI generates a growing share of production code, questions of attribution, bias, security and accountability require empirical grounding. Negri-Ribalta et al. [7] and Xu et al. [8] both identify these as unresolved.

III. LITERATURE REVIEW

Artificial Intelligence has become a significant force in software development, and a substantial body of recent work examines how it affects automation, productivity and error rates relative to manual practice.

Foundational reviews establish the technical landscape. Odeh, Odeh and Mohammed [2] provide a comparative review of AI techniques for automated code generation, evaluating deep learning, machine learning, evolutionary algorithms and NLP against accuracy, efficiency, scalability, correctness and generalisation. Kumar [12] examines AI integration across planning, coding, testing and deployment, highlighting both automation benefits and ethical obligations. Upadhyaya [11] surveys the role of AI across the development lifecycle, noting improvements in efficiency alongside persistent interpretability challenges. Nyaga [13] contributes a systematic review of machine learning's impact on code generation, error detection and program maintenance, identifying model interpretability and workflow integration as unresolved gaps. Alenezi and Akour [9] review current practices and future directions, and Khade and Sambhe [10] analyse AI's effect on code generation, testing, maintenance and security, discussing tools such as Codex and Copilot while flagging ethical and security concerns.

A second group of studies addresses generative AI specifically. Banh, Holldack and Strobel [1], publishing in Information and Software Technology, examine how generative AI transforms software engineering, documenting shifts in optimisation and manual workload. Jackson et al. [6] set out a research agenda on creativity, generative AI and software development, focusing on how human-AI collaboration affects developer creativity. Qiu, Puccinelli, Ciniselli and Di Grazia [14] project the transformation of the developer's routine towards 2030, emphasising the changing nature of engineering roles under automation. Zakharov, Koshchenko and Sergeyuk [15] study perceived roles of AI in software engineering and their effect on adoption, identifying both productivity gains and verification difficulties.

Correctness and security form a third strand. Cotroneo et al. [3] propose ACCA, a fully automated method using symbolic execution to assess the correctness of AI-generated code in security contexts. Negri-Ribalta, Geraud-Stewart, Sergeeva and Lenzini [7] conduct a systematic literature review on the impact of AI models on the security of code generation, recommending practices to strengthen AI-assisted coding. Kharma, Choi, Alkhanafseh and Mohaisen [4] analyse security and quality in LLM-generated code across multiple languages and models, proposing mitigation techniques. Abbassi, Da Silva, Nikanjam and Khomh [16] construct a taxonomy of inefficiencies in LLM-generated Python code and propose optimisation strategies for maintainability and performance.

Advances in generation technique constitute a fourth strand. Wong and Tan [19] study the alignment of crowd-sourced human feedback for reinforcement learning on code generation, improving output reliability. Li et al. [21] present CodeTree, an agent-guided tree search framework that refines generation through structured search and self-correction. Ashrafi, Bouktif and Mediani [17] evaluate multi-agent collaboration and runtime debugging, reporting improvements in accuracy, reliability and latency. Dong et al. [20] survey code generation with LLM-based agents, characterising autonomy, workflow decomposition and tool integration across the lifecycle. Oertel, Klünder and Hebig [18] address a practical question directly relevant to adoption, examining how many GitHub Copilot suggestions a developer should evaluate before settling on one.

Testing and quality assurance form a fifth strand. Job [24] examines automating and optimising software testing using AI techniques. Ramadan, Yasin and Pektaş [22] review the role of AI and machine learning in software testing, reporting improvements in defect prediction and quality assurance. Maarleveld et al. [23] provide a systematic mapping study on graph machine learning for static source code analysis.

Finally, methodological and ethical contributions frame the field. Hymel [26] proposes the AI-Native Software Development Lifecycle, a methodology in which AI is integral rather than additive. Terragni, Vella, Roop and Blincoe [5] present a vision of AI-driven software engineering spanning requirements, design and continuous development. Xu, Li, Li and Tan [8] define ethically sourced code generation, addressing data licensing, fairness and transparency. Narvekar, Maurya, Parab and Prasad [25] describe CODE-GENIE, an applied AI-powered code generation model.

Taken together, this literature documents a clear shift from manual coding towards AI-assisted development. The consensus is that AI accelerates the development lifecycle and improves productivity on routine tasks. Notably, however, the reviewed studies concentrate on technical capability, tool benchmarking and correctness verification. Comparatively few examine developer-reported outcomes such as satisfaction, trust or compensation at population scale, and fewer still control for the confounding factors that shape those outcomes. The present study addresses that specific omission.

IV. GAP ANALYSIS

Twenty-six studies were reviewed in detail. Table I summarises each contribution and the gap it leaves unaddressed. A consistent pattern emerges: the literature is overwhelmingly technical, concentrating on model capability, benchmark performance and correctness verification. Developer-reported outcomes at population scale, and the confounding factors that shape them, are largely absent.

TABLE I. GAP ANALYSIS OF REVIEWED LITERATURE

Sr.	Author (Year)	Area of Study	Contribution and Identified Gap
1	Odeh, Odeh & Mohammed [2]	Automated code generation; deep learning, evolutionary algorithms, ML, NLP	Comparative review of AI code-generation techniques assessed for efficiency and scalability. Gap: evaluation is technical; developer-reported outcomes are not examined.
2	Kumar [12]	AI across the software development lifecycle	Examines AI in planning, coding, testing and deployment with attention to ethics. Gap: no empirical measurement of outcomes at population scale.
3	Upadhyaya [11]	AI and chatbots in the development lifecycle	Surveys AI automation across the SDLC and developer productivity. Gap: interpretability challenges noted but not quantified.
4	Nyaga [13]	ML in code generation, error detection and maintenance	Systematic review of ML contributions to generation, bug detection and maintenance. Gap: model interpretability and workflow integration remain unaddressed.
5	Alenezi & Akour [9]	AI-driven innovation in software engineering	Reviews emerging AI techniques and automation benefits. Gap: ethical and reliability challenges identified but not empirically tested.
6	Khade & Sambhe [10]	Code generation, testing, maintenance, security	Analyses AI's effect on coding, testing and cybersecurity using Codex and Copilot. Gap: tool-centric; no comparison against manual practice.
7	Banh, Holldack & Strobel [1]	Generative AI in software engineering	Documents how generative AI transforms workflows and reduces manual workload. Gap: organisational rather than individual-outcome focus.
8	Cotroneo et al. [3]	Correctness of AI-generated code	Proposes ACCA, an automated symbolic-execution method for assessing correctness in security contexts. Gap: verification treated as a technical problem, not an adoption barrier.
9	Negri-Ribalta et al. [7]	Security of AI code generation	Systematic review of AI models' impact on code security with recommended practices. Gap: developer trust not measured.
10	Terragni, Vella, Roop & Blincoe [5]	Future of AI-driven software engineering	Vision spanning requirements, design and continuous development. Gap: forward-looking; not grounded in current practitioner data.
11	Jackson et al. [6]	Creativity and human-AI collaboration	Research agenda on generative AI and developer creativity. Gap: agenda-setting rather than empirical.
12	Ashrafi, Bouktif & Mediani [17]	Multi-agent collaboration and runtime debugging	Evaluates multi-agent LLM systems for accuracy, reliability and latency. Gap: benchmark-based; no field data.
13	Kharma et al. [4]	Security and quality of LLM-generated code	Multi-language, multi-model vulnerability analysis with mitigation techniques. Gap: does not connect security findings to adoption behaviour.
14	Wong & Tan [19]	Reinforcement learning with human feedback	Aligns crowd-sourced feedback to improve generation reliability. Gap: model-level; user outcomes not assessed.
15	Xu, Li, Li & Tan [8]	Ethically sourced code generation	Defines ethical sourcing, licensing and fairness in generated code. Gap: normative framework without practitioner validation.
16	Abbassi et al. [16]	Inefficiencies in LLM-generated code	Taxonomy of inefficiencies in generated Python with optimisation strategies. Gap: maintainability cost not weighed against productivity gain.
17	Qiu et al. [14]	Developer experience towards 2030	Projects the transformation of developer routine under AI assistance. Gap: projective rather than measured.
18	Zakharov, Koshchenko & Sergeyuk [15]	Perceived roles of AI and adoption	Studies perceived AI roles and their effect on adoption in IDEs. Gap: perception studied without linkage to satisfaction or compensation.
19	Dong et al. [20]	LLM-based code generation agents	Surveys autonomy, decomposition and tool integration across the SDLC. Gap: capability survey; no outcome evaluation.
20	Oertel, Klünder & Hebig [18]	GitHub Copilot suggestion evaluation	Examines how many Copilot suggestions should be checked before selection. Gap: narrow task focus.
21	Li et al. [21]	Agent-guided code generation (CodeTree)	Structured tree search with self-correction for improved generation. Gap: benchmark performance only.
22	Hymel [26]	AI-Native SDLC methodology	Proposes a lifecycle in which AI is integral rather than additive. Gap: methodological proposal without empirical validation.
23	Job [24]	AI in software testing	Automating and optimising testing using AI techniques. Gap: predates current generative tooling.
24	Ramadan, Yasin & Pektaş [22]	AI and ML in software testing	Reviews defect prediction and quality assurance improvements. Gap: testing-specific; lifecycle-wide outcomes not covered.
25	Maarleveld et al. [23]	Graph ML for static source code analysis	Systematic mapping of graph machine learning applied to code analysis. Gap: technique-focused.
26	Narvekar et al. [25]	Applied AI code generation (CODE-GENIE)	Presents an AI-powered code generation model. Gap: single-tool case study.

The gap addressed here is therefore specific. Prior work establishes that AI can generate, test and repair code, and characterises the risks of doing so. What remains untested is whether adoption produces measurable improvement in developer-level outcomes once confounding factors are controlled.

V. OBJECTIVES, VARIABLES AND HYPOTHESES

A. *Objectives of the Study*

To study the impact of Artificial Intelligence on modern application development processes.
To compare AI-assisted coding techniques with traditional manual coding approaches.
To evaluate the efficiency, accuracy and productivity achieved through AI-based development tools.
To understand developer perception and adaptability towards AI-integrated development environments.
To identify the challenges and limitations encountered when implementing AI in software engineering practice.

B. *Identification of Variables*

Table II sets out the variables analysed and the dataset field from which each was operationalised.

TABLE II. IDENTIFICATION OF VARIABLES

Type	Variable	Operational Definition (dataset field)
Independent	Adoption of AI tools	Whether the respondent currently uses AI tools in development (AISelect)
Dependent	Perceived productivity	Reported benefits of AI use (AIBen)
Dependent	Job satisfaction	Overall satisfaction, 0-10 scale (JobSat); nine-item battery (JobSatPoints)
Dependent	Perceived accuracy	Trust in accuracy of AI-generated code (AIAcc)
Dependent	Compensation	Annual compensation normalised to USD (ConvertedCompYearly)
Control	Country	Respondent's country of residence (Country)
Control	Professional experience	Years of professional coding experience (YearsCodePro)
Control	Developer role	Primary developer role (DevType)
Control	Organisation size	Number of employees (OrgSize)
Qualitative	Sentiment	Overall attitude towards AI (AISent)
Qualitative	Ethical concerns	Ethical issues identified by respondent (AIEthics)
Qualitative	Challenges	Barriers reported in AI adoption (AIChallenges)

C. *Hypotheses*

Seven hypotheses were formulated, each stated in null and alternative form and tested at the 5% level of significance.
H1. Productivity. H0: AI-assisted development does not significantly improve developer productivity compared with manual coding. H1: AI-assisted development leads to significantly higher developer productivity.
H2. Job satisfaction. H0: There is no significant difference in job satisfaction between developers who use AI tools and those who code manually. H1: Developers using AI tools report significantly higher job satisfaction.
H3. Efficiency and accuracy. H0: AI tools do not significantly affect perceived coding efficiency or accuracy compared with manual coding. H1: AI-assisted environments significantly improve perceived efficiency and accuracy.
H4. Compensation. H0: AI adoption has no association with developer compensation levels. H1: Developers who adopt AI tools exhibit higher compensation.
H5. Challenges. H0: AI adoption does not introduce significant challenges relating to trust, contextual understanding or security. H1: AI adoption introduces notable challenges of trust, contextual accuracy and data security.
H6. Overall impact. H0: Integrating AI into software development does not significantly influence developer sentiment or workflow outcomes. H1: AI integration significantly affects sentiment and workflow outcomes.
H7. Reliability. H0: The job-satisfaction measurement scale does not demonstrate acceptable internal consistency. H1: The scale demonstrates high reliability as measured by Cronbach's alpha.

VI. RESEARCH METHODOLOGY

A. *Research Design*

This study adopts a descriptive and comparative research design based on secondary data. The analysis was conducted between August 2025 and August 2026. A quantitative approach was selected because the research questions concern measurable differences between two populations of developers, those who use AI tools and those who do not, across outcomes that are recorded numerically or categorically.

Secondary analysis was chosen in preference to primary data collection for two reasons. First, the scale required to detect population-level differences in compensation and satisfaction is far beyond what an individual student researcher could collect. Second, an independently administered, publicly documented instrument reduces the risk of researcher bias in question design.

B. Data Collection

Data were obtained from the Stack Overflow Annual Developer Survey 2024 [27], a publicly available dataset released under an Open Database License and downloadable from survey.stackoverflow.co. The survey was fielded online from 19 May 2024 to 20 June 2024 and yielded 65,437 responses from developers in 185 countries. Respondents were recruited through Stack Overflow's own channels, principally onsite messaging, blog posts, email newsletters, banner advertisements and social media. Sampling is therefore non-probability and self-selected, a characteristic addressed in Section 11.

The dataset was selected because it contains, in a single instrument, the variables required by this study: AI tool adoption (AISelect), sentiment towards AI (AISent), reported benefits (AIBen), trust in AI accuracy (AIAcc), reported challenges (AIChallenges), expectations of future integration (AINext), job satisfaction on a 0-10 scale (JobSat) together with a nine-item satisfaction battery (JobSatPoints), and annual compensation normalised to United States dollars (ConvertedCompYearly), alongside demographic controls for country, developer role, organisation size and years of professional experience.

No primary data were collected. All figures reported in this paper were computed directly from the raw response file rather than taken from any published summary.

C. Sample Size Determination

Because the target population of professional software developers is effectively infinite, Cochran's formula for large populations was applied to establish the minimum sample required:

n = (Z^2 x p x (1 - p)) / e^2

where Z = 1.96 at the 95% confidence level, p = 0.5 representing maximum variability as the safest assumption, and e = 0.05 representing a 5% margin of error. Substituting these values gives n = (1.96^2 x 0.5 x 0.5) / 0.05^2 = 384.16, that is, a required minimum of 385 respondents.

The achieved sample of 65,437 responses exceeds this minimum by a factor of approximately 170. Sub-group analyses reported in this paper retain sample sizes well above the threshold: the job-satisfaction comparison draws on 18,233 AI users and 6,784 manual coders, and the compensation comparison on 14,777 and 5,562 respectively. The sample is therefore adequate for all inferential tests performed.

D. Tools and Techniques of Analysis

Analysis was performed in Python 3.14 using the pandas library for data handling, SciPy for statistical testing and Matplotlib for visualisation. Non-parametric tests were selected throughout because the principal dependent variables are ordinal (job satisfaction on a 0-10 scale) or heavily right-skewed (annual compensation), violating the normality assumption of parametric alternatives.

The Mann-Whitney U test was used for two-group comparisons of ordinal and skewed continuous variables. The chi-square test of independence was used to assess association between categorical variables, with Cramer's V reported as the effect-size measure. The chi-square goodness-of-fit test was used to assess whether categorical distributions departed from uniformity. Cohen's d was computed for continuous comparisons to distinguish statistical significance from practical significance, a distinction that proves material given the sample size. Welch's t-test is reported alongside Mann-Whitney U for the job-satisfaction comparison as a robustness check.

Compensation records were restricted to values greater than zero and below USD 1,000,000 to exclude data-entry artefacts. Records with missing values on a given variable were excluded pairwise rather than listwise, and the effective sample size is reported alongside each result.

E. Reliability of the Measurement Instrument

The internal consistency of the job-satisfaction construct was assessed using Cronbach's alpha across the nine available JobSatPoints items (JobSatPoints_1 and JobSatPoints_4 to JobSatPoints_11; items 2 and 3 are not present in the released dataset). Across 29,095 complete cases the coefficient obtained was alpha = 0.931.

A value above 0.9 indicates excellent internal consistency, confirming that respondents interpreted the constituent items uniformly and that the scale is suitable for the comparative analysis that follows. This result also constitutes the test of H7.

VII. DATA ANALYSIS AND FINDINGS

A. Description of the Dataset

The dataset comprises 65,437 responses. Unlike the descriptive summaries frequently circulated from this survey, every statistic reported below was recomputed from the raw file, and the denominator used is stated explicitly in each case. This matters because the survey contains substantial item non-response: 6.9% of respondents did not answer the AI adoption question and 29.9% did not answer the sentiment question. Percentages computed over all rows and percentages computed over valid responses therefore differ materially, and conflating the two is a common source of error in secondary analyses of this instrument.

B. Respondent Profile

The respondent base is examined first, since the composition of the sample conditions every subsequent comparison.

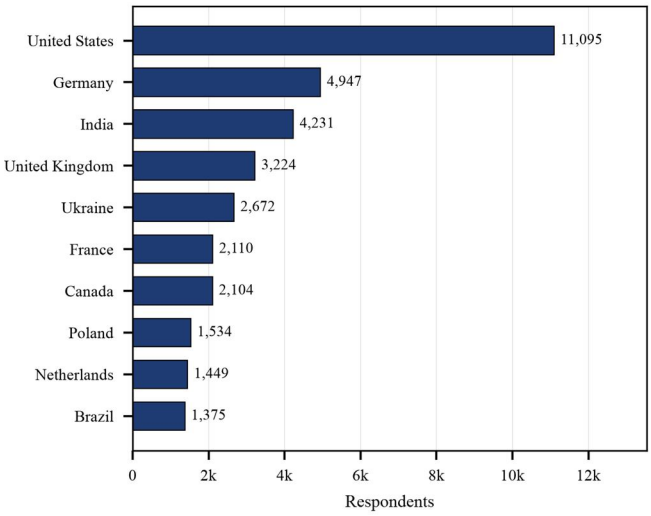


Fig. 1 Top ten countries of respondents

The United States contributes 11,095 respondents, followed by Germany (4,947), India (4,231) and the United Kingdom (3,224). This geographic concentration proves consequential for the compensation analysis in Section VII-E.

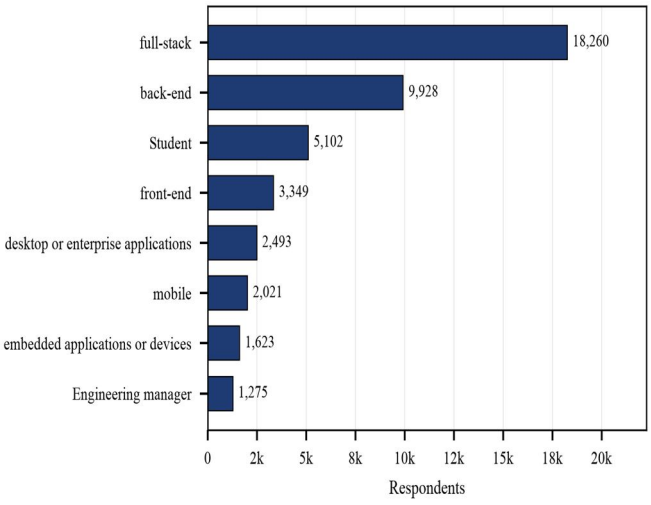


Fig. 2 Principal developer roles

Full-stack developers form the largest group at 18,260, followed by back-end developers (9,928) and students (5,102). These roles span the coding, testing and deployment stages at which AI tools are principally applied.

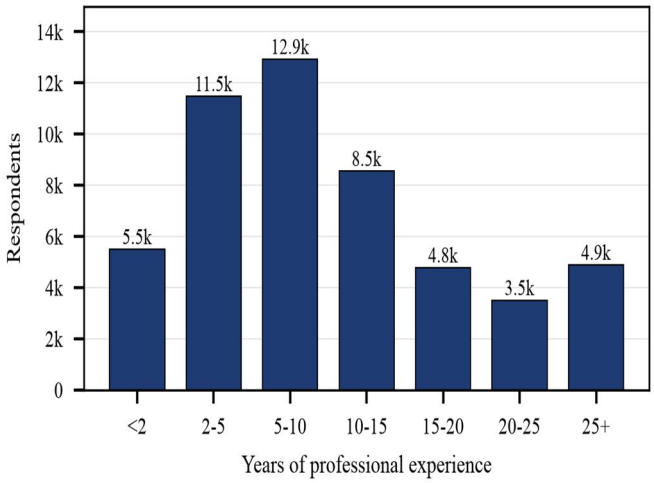


Fig. 3 Professional coding experience

The median respondent has 7 years of professional experience; 57.9% report fewer than 10 years and 13.2% more than 20. The sample skews towards early- and mid-career developers, which also bears on the compensation analysis.

C. AI Adoption and Patterns of Use

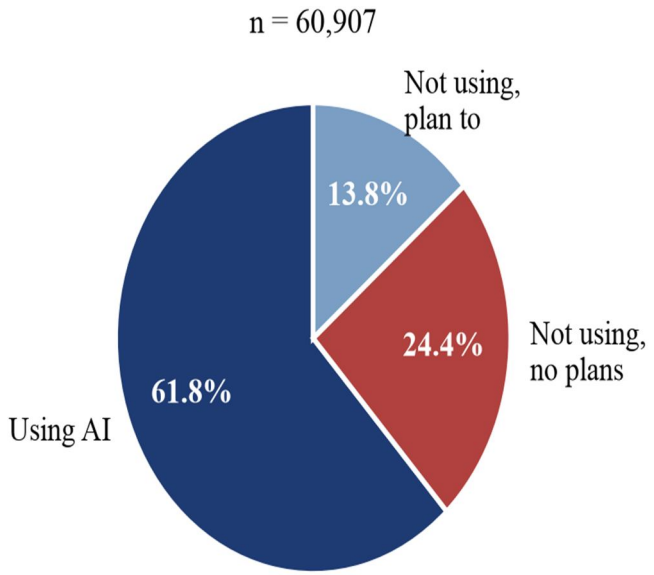


Fig. 4 Adoption of AI tools among developers

Of 60,907 respondents answering, 57.6% of all respondents (61.8% of valid answers) currently use AI tools. A further 12.8% intend to adopt shortly, while 22.7% neither use them nor plan to.

Respondents who plan to adopt shortly are frequently grouped with manual coders in secondary analyses. That grouping is misleading, since their stated intention places them closer to adopters. All comparisons here contrast current AI users against those who neither use AI nor plan to, excluding the intermediate group.

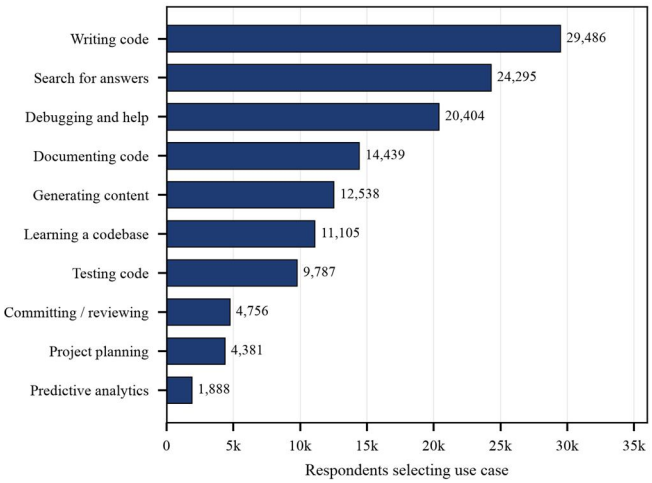


Fig. 5 Principal use cases for AI in development

Writing code dominates at 29,486 respondents, followed by searching for answers (24,295) and debugging (20,404). Tasks requiring reasoning about the system as a whole, such as project planning (4,381) and predictive analytics (1,888), remain rare. AI is currently applied to localised, well-bounded tasks rather than architectural work.

D. Developer Sentiment and Expectations

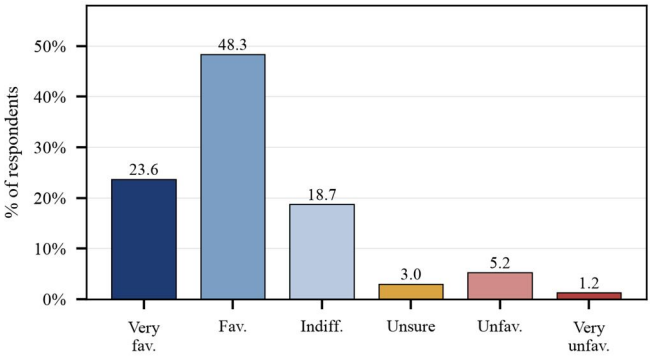


Fig. 6 Developer sentiment towards AI

Among 45,873 respondents expressing a view, 48.3% are favourable and 23.6% very favourable, a combined 72.0%. Indifference accounts for 18.7%, negative views for 6.4% and uncertainty for 3.0%.

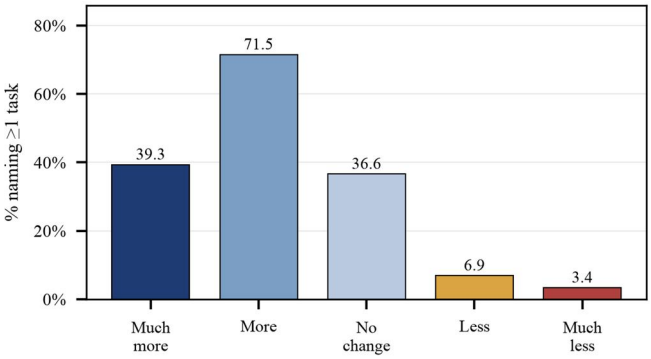


Fig. 7 Expectations for future AI integration

Of those responding, 71.5% expect at least one task to become more integrated with AI and 39.3% much more integrated, against 6.9% anticipating reduced integration.

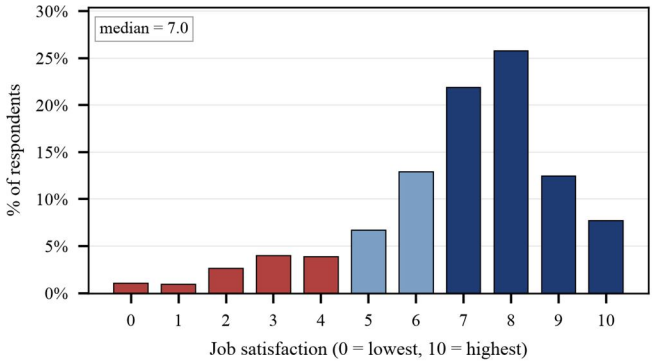


Fig. 8 Distribution of overall job satisfaction

Job satisfaction is left-skewed with a median of 7.0. Ratings of 7 and 8 account for 47.7% of responses, and ratings below 5 for 12.6%. This establishes the baseline for the group comparison that follows.

D. Objective-wise Analysis

Each objective is addressed in turn, with the corresponding hypothesis test reported in Section VII-F.

1) Objective 1: Impact of AI on Development Processes:

Figures 5 and 7 together address this objective. AI is embedded principally in code authoring, search and debugging, and developers expect that integration to deepen. AI has become a routine component of development rather than an experimental adjunct.

2) Objective 2: AI-Assisted versus Manual Coding:

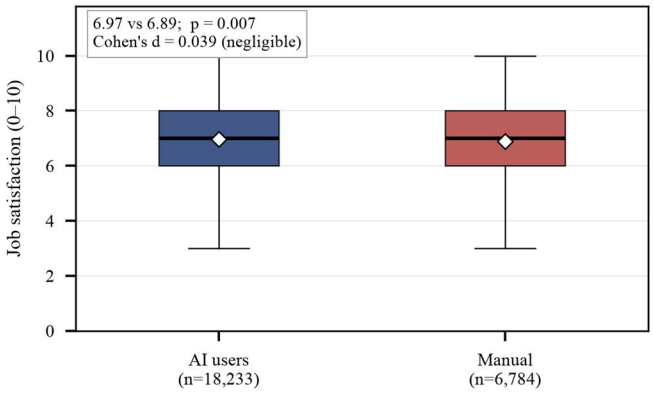


Fig. 9 Job satisfaction, AI users compared with manual coders

This comparison requires care, because it is frequently reported in a form the data do not support. AI users report mean job satisfaction of 6.97 (n = 18,233) against 6.89 for manual coders (n = 6,784). The medians are identical at 7.0.

The difference is statistically significant (Mann-Whitney U = 63,049,886, p = 0.016; Welch t = 2.698, p = 0.007), but negligible in magnitude: Cohen's d = 0.039, well below the 0.2 threshold for a small effect. With samples exceeding 25,000, statistical significance is achievable for differences of no practical consequence. AI adoption is associated with no meaningful difference in job satisfaction.

3) Objective 3: Efficiency, Accuracy and Productivity:

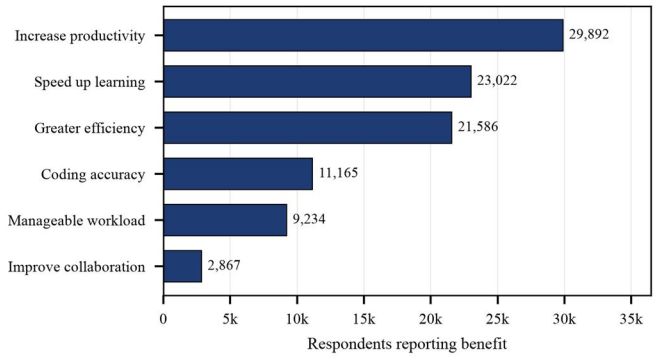


Fig. 10 Reported benefits of AI adoption

Increased productivity is the most cited benefit at 29,892 respondents, representing 81.0% of adopters who answered. Accelerated learning follows at 23,022 and greater efficiency at 21,586. Improved coding accuracy is cited by only 11,165.

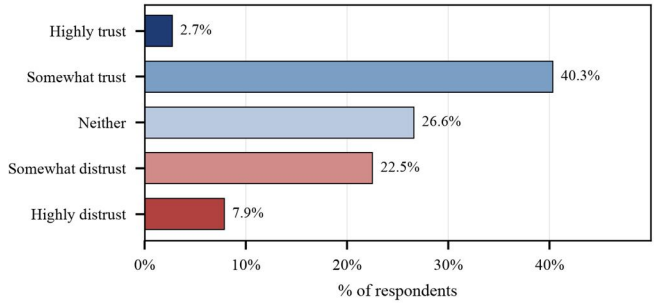


Fig. 11 Trust in the accuracy of AI-generated code

Trust is more equivocal. Of 37,302 respondents, 40.3% somewhat trust AI-generated code and only 2.7% highly trust it, a combined 43.0%. Against this, 30.4% distrust it and 26.6% are neutral. Developers value AI output while withholding full confidence, consistent with the correctness concerns documented by Cotroneo et al. [3].

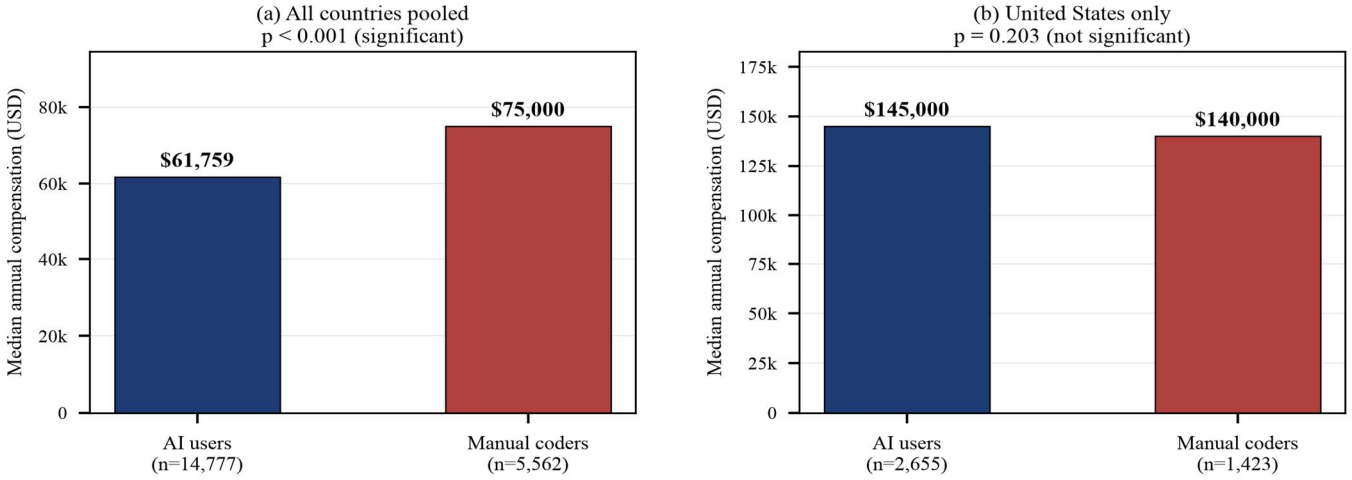


Fig. 12 Median annual compensation, pooled and controlled for country

The compensation comparison produces the study's most consequential finding, and it contradicts the expected direction.
Panel (a) pools all countries. Manual coders report a median of USD 75,000 (n = 5,562) against USD 61,759 for AI users (n = 14,777), highly significant (Mann-Whitney U = 34,939,784, p < 0.001) and favouring manual coders, the opposite of the hypothesis. This should not be accepted at face value. AI adoption reaches 67.9% in India and 72.3% in Ukraine, against 50.7% in the United Kingdom and 54.2% in the United States. Since compensation varies enormously between these economies, pooling produces a spurious association driven by national income rather than AI use.

Panel (b) removes the country effect. Within the United States alone, AI users report a median of USD 145,000 (n = 2,655) against USD 140,000 for manual coders (n = 1,423), not statistically significant (U = 1,934,622, p = 0.203). A secondary confound reinforces this: AI users have a median of 6 years of professional experience against 10 for manual coders.

The pooled result is an instance of Simpson's paradox, in which an association reverses when a confounding variable is controlled. The data provide no evidence that AI adoption affects developer compensation in either direction.

4) Objective 4: Perception and Adaptability:

Figure 6 established that 72.0% hold favourable views. The association between adoption and sentiment is significant (chi-square = 3,144.1, df = 5, $p < 0.001$) with Cramer's V = 0.262, a moderate rather than strong association.

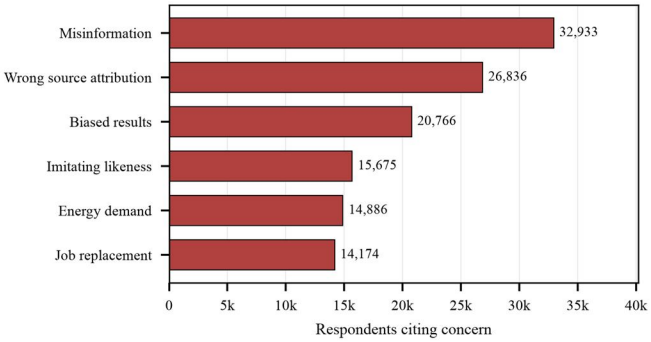


Fig. 13 Ethical concerns reported by developers

Leading concerns are circulating misinformation (32,933), missing or incorrect source attribution (26,836) and biased results (20,766). Job displacement ranks last of the six at 14,174, which sits at odds with much public commentary.

5) Objective 5: Challenges and Limitations

Fig. 14 Reported challenges in AI adoption

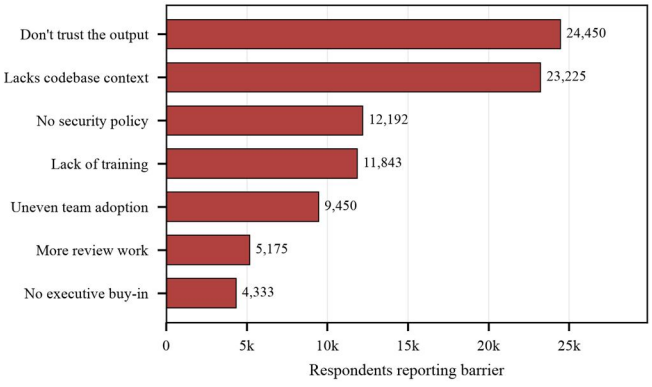


Fig. 15 Principal barriers to AI adoption

Of 37,531 respondents reporting challenges, 65.1% cite distrust of AI output and 61.9% report that AI tools lack context of their codebase or internal architecture. Absent security policy is reported by 32.5% and inadequate training by 31.6%.

The leading barriers are not concerns about capability but about verification and context. Developers do not doubt that AI can produce code; they doubt they can trust the code produced, and observe that the tools do not understand the systems into which it must fit.

E. Hypothesis Testing Results

Table III reports the outcome of each hypothesis test, conducted at the 5% level using non-parametric methods appropriate to ordinal and skewed data.

TABLE III. SUMMARY OF HYPOTHESIS TESTING RESULTS (A = 0.05)

Hyp.	Focus	Test Applied	Statistic	p-value	Decision
H1	Productivity	Binomial test of proportion	81.0% of adopters; z = 119.17	< 0.001	Reject H0
H2	Job satisfaction	Mann-Whitney U	U = 63,049,886; d = 0.039	0.016	Reject H0 (effect negligible)
H3	Efficiency / accuracy	Chi-square goodness-of-fit	chi-square = 1,661.4, df = 2	< 0.001	Reject H0
H4	Compensation (pooled)	Mann-Whitney U	U = 34,939,784	< 0.001	Reject H0 (direction reversed)
H4a	Compensation (US only)	Mann-Whitney U	U = 1,934,622	0.203	Fail to reject H0
H5	Challenges	Binomial test of proportion	65.1% cite distrust of output	< 0.001	Reject H0
H6	Sentiment and impact	Chi-square test of independence	chi-square = 3,144.1, df = 5; V = 0.262	< 0.001	Reject H0
H7	Scale reliability	Cronbach's alpha	alpha = 0.931 (k = 9, n = 29,095)	-	Reject H0

Two entries merit comment. H2 is rejected formally, yet the effect size of 0.039 renders the difference practically meaningless; both results are reported rather than the favourable one alone. H4 is reported in two parts: the pooled test rejects the null but in the direction opposite to the hypothesis, while the country-controlled test fails to reject it. The controlled result is the one that supports interpretation.

VIII. RESULTS AND DISCUSSION

The results divide into findings that confirm prevailing expectations and findings that contradict them. Both are reported.

Three findings confirm the conventional account. First, adoption is majority practice: 57.6% of all respondents, and 61.8% of those answering the question, currently use AI tools in development. Second, productivity is the dominant reported benefit, cited by 81.0% of adopters, well ahead of accelerated learning (62.4%) and general efficiency (58.5%). Third, sentiment is strongly favourable, with 72.0% of respondents expressing favourable or very favourable views and a significant association between adoption and sentiment (chi-square = 3144.1, df = 5, $p < 0.001$, Cramer's V = 0.262).

Two findings contradict the conventional account, and both concern claims that are widely repeated without testing.

The job-satisfaction advantage is real but negligible. AI users report mean satisfaction of 6.97 against 6.89 for manual coders. The difference is statistically significant (Mann-Whitney U = 63,049,886, $p = 0.016$; Welch t = 2.698, $p = 0.007$), but Cohen's d is 0.039, far below the 0.2 threshold conventionally regarded as a small effect. The medians of both groups are identical at 7.0. With sample sizes above 25,000, statistical significance is attainable for differences of no practical consequence, and this is such a case. The honest conclusion is that AI adoption is not associated with any meaningful improvement in job satisfaction.

The compensation advantage does not exist. Pooled across all countries, manual coders report a median annual compensation of USD 75,000 against USD 61,759 for AI users, a difference that is highly significant (Mann-Whitney U = 34,939,784, $p < 0.001$) and in the opposite direction to the hypothesis. Rather than accept this at face value, the relationship was examined for confounding. AI adoption varies substantially by country, reaching 67.9% in India and 72.3% in Ukraine against 50.7% in the United Kingdom and 54.2% in the United States. Because compensation also varies enormously by country, pooling produces a spurious association. Restricting the comparison to United States respondents alone removes the country effect: AI users report a median of USD 145,000 against USD 140,000 for manual coders, a difference that is not statistically significant ($p = 0.203$). AI users are also earlier in their careers, with a median of 6 years of professional experience against 10 for manual coders, which compounds the effect.

This is a textbook instance of Simpson's paradox, in which an association present in aggregated data reverses or disappears when the data are disaggregated by a confounding variable. The correct conclusion is that the survey provides no evidence that AI adoption raises developer compensation, and equally no evidence that it lowers it.

Trust remains the binding constraint on adoption. Although 43.0% of respondents somewhat or highly trust the accuracy of AI-generated code, 30.4% actively distrust it and only 2.7% report high trust. Among reported challenges, 65.1% cite distrust of output and 61.9% report that AI tools lack context of their codebase and internal architecture. These two barriers dominate all others, including security policy gaps (32.5%) and inadequate training (31.6%). The pattern is consistent with the correctness and security literature reviewed in Section 3, particularly Cotroneo et al. [3] and Kharma et al. [4], and indicates that the constraint on deeper integration is verification rather than capability.

IX. KEY ISSUES IDENTIFIED

The analysis surfaces five issues that limit the effective integration of AI into professional development practice.

Trust and reliability deficits. Distrust of AI output is the single most frequently cited barrier, reported by 65.1% of respondents. Only 2.7% express high trust in the accuracy of generated code, indicating that acceptance is provisional rather than settled.

Loss of context. 61.9% of respondents report that AI tools lack knowledge of their codebase, internal architecture or organisational conventions. This is a structural limitation rather than a matter of model quality, and it constrains AI to localised rather than architectural tasks. Organisational readiness and governance. 32.5% of respondents report that their organisation lacks policies to manage security risk arising from AI use, and 31.6% report inadequate training. Adoption is therefore proceeding ahead of the governance required to support it.

Ethical concerns. Respondents identify circulating misinformation, missing or incorrect attribution of data sources, and biased results as the leading ethical risks, ahead of energy consumption and employment displacement.

Absence of demonstrated outcome gains. The most significant issue identified by this study is that the measurable benefits of AI adoption are narrower than commonly claimed. Productivity gains are self-reported and widely endorsed, but neither job satisfaction nor compensation shows a defensible advantage once confounding is addressed.

X. RECOMMENDATIONS

Five recommendations follow from the findings.

Invest in verification rather than generation. Since distrust of output is the principal barrier, the greatest marginal return lies in tooling that helps developers verify AI-generated code, including automated correctness assessment of the kind proposed by Cotroneo et al. [3], rather than in further improvements to generation capability.

Prioritise context integration. Because 61.9% of respondents identify missing codebase context as a limitation, organisations should favour tools capable of indexing internal repositories, architectural conventions and organisational knowledge over general-purpose assistants.

Establish governance before scaling adoption. Given that roughly one third of respondents report absent security policies, organisations should define data-handling protocols, review checkpoints for AI-generated code, provenance tracking and accountability mechanisms prior to broad rollout.

Adopt hybrid development models deliberately. AI should be assigned routine generation, documentation and test-scaffolding work, while architectural design, security decisions and final verification remain with human developers. This allocation reflects both the measured productivity benefit and the measured trust deficit.

Base adoption decisions on measured outcomes. Organisations and researchers should evaluate AI tools against verifiable indicators rather than assumed benefits. This study demonstrates that two widely cited advantages, satisfaction and compensation, do not withstand controlled examination, and similar caution should be applied to other claims.

XI. LIMITATIONS OF THE STUDY

Five limitations should be borne in mind when interpreting these results.

The design is cross-sectional and correlational. The data capture a single point in time and cannot establish causation. Where associations are reported, the direction of causality is not determined; developers who are already satisfied may be more inclined to adopt AI tools rather than the reverse.

Key measures are self-reported. Productivity, efficiency and accuracy are reported by respondents rather than observed. No independent measurement of code quality, defect density or delivery speed is available in the dataset, so reported productivity gains reflect perception rather than measured output.

The sample is not globally representative. Recruitment was non-probability and self-selected: respondents were reached through Stack Overflow's own channels, so the sample over-represents developers already engaged with that platform and skews towards the United States, Europe and English-speaking developers. Regions and development cultures outside that orbit are under-represented. Results describe this population and should not be generalised to all developers without qualification.

Item non-response is substantial. Between 6.9% and 43.0% of responses are missing depending on the variable, and non-response is unlikely to be random. Pairwise deletion was used, with effective sample sizes reported throughout, but residual non-response bias cannot be excluded.

Controls are partial. The compensation analysis controls for country and reports experience differences, but does not construct a full multivariate model incorporating role, organisation size, education and seniority simultaneously. The conclusion that no compensation effect exists is therefore robust to the principal confounder identified but not to all possible confounders.

XII. SCOPE FOR FUTURE RESEARCH

Four directions follow directly from the limitations above.

Longitudinal measurement. Tracking the same developers across successive survey waves would permit examination of whether outcomes change following adoption, addressing the causal ambiguity inherent in cross-sectional data.

Objective productivity measurement. Pairing survey responses with repository telemetry such as commit frequency, defect rates, review cycle time and rework would allow perceived productivity to be tested against measured productivity.

Full multivariate modelling. Regression or matched-sample analysis incorporating country, experience, role, organisation size and education simultaneously would establish whether any compensation effect survives comprehensive control.

Verification tooling. Given that trust is the binding constraint identified here, research into practical verification of AI-generated code, extending the direction set out by Cotroneo et al. [3] and Negri-Ribalta et al. [7], addresses the barrier that adoption data identify as most consequential.

Indian and emerging-market context. Adoption in India is markedly higher than the global average at 67.9%, yet the developer experience in emerging markets is under-studied. Focused research on this context, including domains such as e-governance, would fill a genuine gap.

XIII. CONCLUSION

This study set out to compare AI-assisted and manual application development using large-scale secondary data, and to test rather than assume the benefits attributed to AI. Analysis of 65,437 responses from the Stack Overflow Annual Developer Survey 2024, with a measurement instrument of demonstrated reliability (Cronbach's alpha = 0.931), supports three conclusions.

AI adoption is now majority practice and is perceived as productive. A majority of developers use AI tools, 81.0% of adopters report increased productivity, and 72.0% view AI favourably. The direction of travel is not in doubt.

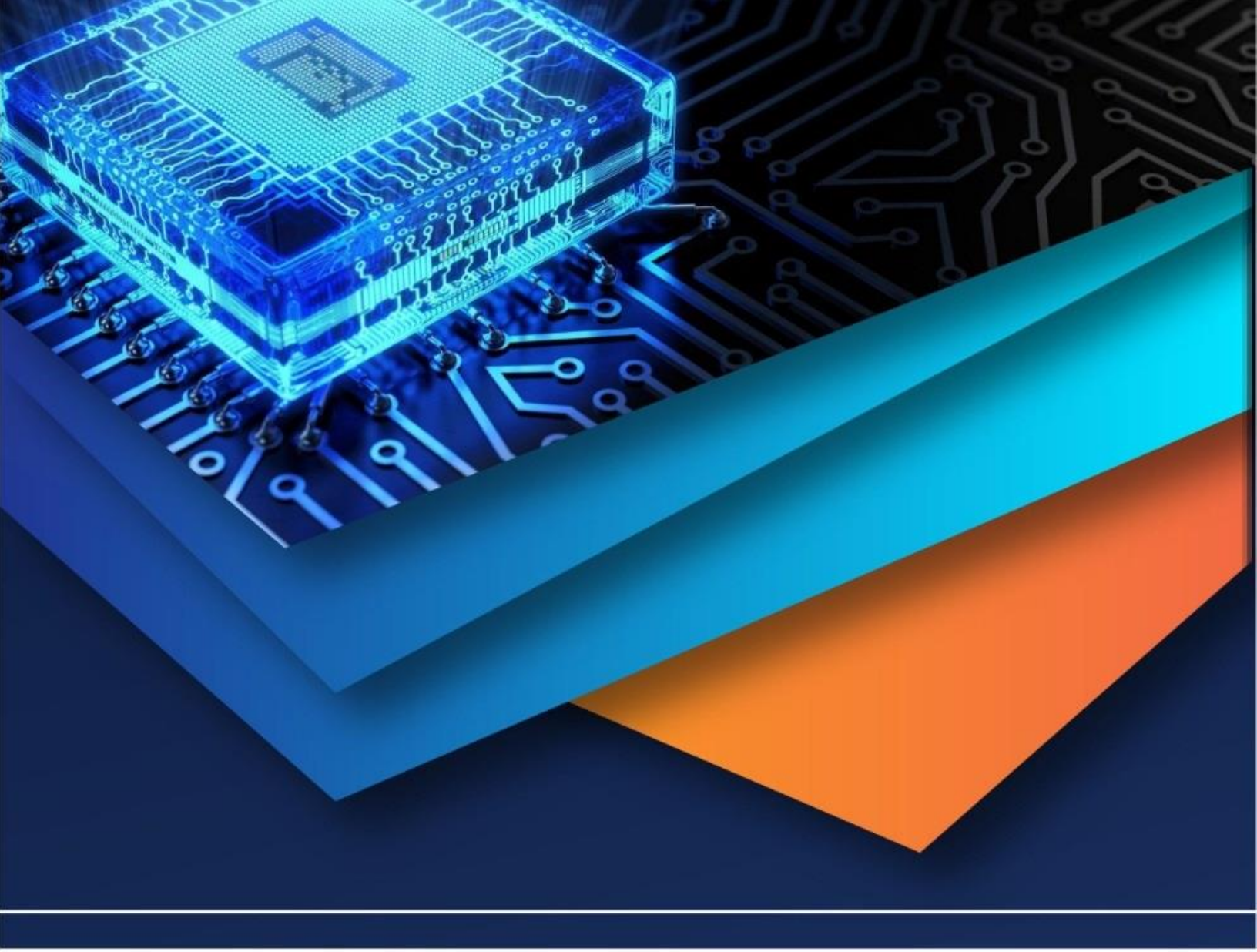
The benefits are narrower than commonly claimed. Neither of the two outcome advantages tested survived examination. The job-satisfaction difference between AI users and manual coders, while statistically significant, is negligible in magnitude (Cohen's d = 0.039) with identical medians. The apparent compensation advantage reverses in pooled data and disappears entirely when national context is controlled ($p = 0.203$), a clear instance of Simpson's paradox. The evidence supports AI as a productivity aid, not as a driver of improved satisfaction or earnings.

Trust, not capability, is the binding constraint. With 65.1% of respondents distrusting AI output and 61.9% citing absent codebase context, the obstacle to deeper integration is verification rather than generation quality. This finding aligns with the correctness and security literature and indicates where effort is best directed.

The broader implication is methodological as much as substantive. Where evidence about AI is frequently asserted rather than tested, this study demonstrates that two widely repeated claims do not hold under controlled analysis. AI is not replacing human developers, and on the available evidence it is not yet making them measurably more satisfied or better paid. It is making them faster at routine work, which is a real and valuable outcome, and it is doing so within a hybrid model in which human judgement retains responsibility for architecture, security and verification. That model, supported by governance, training and better verification tooling, represents the most defensible direction for application development.

REFERENCES

[1] L. Banh, F. Holldack, and G. Strobel, "Copiloting the future: How generative AI transforms software engineering," Information and Software Technology, vol. 183, art. 107751, 2025.
[2] A. Odeh, N. Odeh, and A. S. Mohammed, "A comparative review of AI techniques for automated code generation in software development: Advancements, challenges, and future directions," TEM Journal, vol. 13, no. 1, pp. 726–739, 2024.
[3] D. Cotroneo, A. Foggia, C. Improta, P. Liguori, and R. Natella, "Automating the correctness assessment of AI-generated code for security contexts," Journal of Systems and Software, vol. 216, art. 112113, 2024.
[4] M. F. Kharma, S. Choi, M. Alkhanafseh, and D. Mohaisen, "Security and quality in LLM-generated code: A multi-language, multi-model analysis," arXiv preprint arXiv:2502.01853, 2025.
[5] V. Terragni, A. Vella, P. Roop, and K. Blincoe, "The future of AI-driven software engineering," ACM Transactions on Software Engineering and Methodology, 2025.
[6] V. Jackson, B. Vasilescu, D. Russo, P. Ralph, M. Izadi, and R. Prikladnicki, "Creativity, generative AI, and software development: A research agenda," arXiv preprint arXiv:2406.01966, 2024.
[7] C. Negri-Ribalta, R. Geraud-Stewart, A. Sergeeva, and G. Lenzini, "A systematic literature review on the impact of AI models on the security of code generation," Frontiers in Big Data, vol. 7, art. 1386720, 2024.
[8] Z. Xu, C. Li, Q. Li, and S. H. Tan, "Defining ethically sourced code generation," arXiv preprint arXiv:2507.19743, 2025.
[9] M. Alenezi and M. Akour, "AI-driven innovations in software engineering: A review of current practices and future directions," Applied Sciences, vol. 15, no. 3, art. 1344, 2025.
[10] P. S. Khade and R. U. Sambhe, "Artificial intelligence in software development: A review of code generation, testing, maintenance and security," International Journal of Current Science Research and Review, vol. 8, no. 4, 2025.
[11] N. Upadhyaya, "The role of artificial intelligence in software development: A literature review," Preprint, 2022.
[12] S. Kumar, "Artificial intelligence in software engineering: A systematic exploration of AI-driven development," International Journal of Innovative Research in Science, Engineering and Technology, vol. 13, no. 6, 2024.
[13] F. Nyaga, "AI-driven software engineering: A systematic review of machine learning's impact and future directions," Preprints.org, 2025.
[14] K. Qiu, N. Puccinelli, M. Ciniselli, and L. Di Grazia, "From today's code to tomorrow's symphony: The AI transformation of developer's routine by 2030," arXiv preprint arXiv:2405.12731, 2024.
[15] I. Zakharov, E. Koshchenko, and A. Sergeyuk, "AI in software engineering: Perceived roles and their impact on adoption," arXiv preprint arXiv:2504.20329, 2025.
[16] A. A. Abbassi, L. Da Silva, A. Nikanjam, and F. Khomh, "A taxonomy of inefficiencies in LLM-generated Python code," arXiv preprint arXiv:2503.06327, 2025.
[17] N. Ashrafi, S. Bouktif, and M. Mediani, "Enhancing LLM code generation: A systematic evaluation of multi-agent collaboration and runtime debugging for improved accuracy, reliability, and latency," arXiv preprint arXiv:2505.02133, 2025.
[18] J. Oertel, J. Klünder, and R. Hebig, "Don't settle for the first! How many GitHub Copilot solutions should you check?" Information and Software Technology, vol. 183, art. 107737, 2025.
[19] M. F. Wong and C. W. Tan, "Aligning crowd-sourced human feedback for reinforcement learning on code generation by large language models," arXiv preprint arXiv:2503.15129, 2025.
[20] Y. Dong, X. Jiang, J. Qian, T. Wang, K. Zhang, Z. Jin, and G. Li, "A survey on code generation with LLM-based agents," arXiv preprint arXiv:2508.00083, 2025.
[21] J. Li, H. Le, Y. Zhou, C. Xiong, S. Savarese, and D. Sahoo, "CodeTree: Agent-guided tree search for code generation with large language models," Univ. of Texas at Austin and Salesforce Research, 2025.
[22] A. Ramadan, H. Yasin, and B. Pektaş, "The role of artificial intelligence and machine learning in software testing," Üsküdar University and Dhofar University, 2024.
[23] J. Maarleveld, Y. Yang, S. Wang, and A. Bacchelli, "A systematic mapping study on graph machine learning for static source code analysis," Information and Software Technology, vol. 183, art. 107722, 2025.
[24] M. A. Job, "Automating and optimizing software testing using artificial intelligence techniques," International Journal of Advanced Computer Science and Applications, vol. 12, no. 5, 2021.
[25] P. Narvekar, R. Maurya, S. Parab, and V. Prasad, "CODE-GENIE: An AI-powered code generation model," International Research Journal of Modernization in Engineering Technology and Science, vol. 7, no. 3, 2025.
[26] C. Hymel, "The AI-native software development lifecycle: A theoretical and practical new methodology," Crowdbotics, 2024.
[27] Stack Overflow, "Stack Overflow annual developer survey 2024," [Data set], 2024. [Online]. Available: https://survey.stackoverflow.co/2024/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY