

An empirical analysis of generative artificial intelligence models for enhancing software development efficiency

Halyna Hubal^{1*}, Vladyslav Khambir², Andrii Volivach³, Maryna Kryvoverets⁴, Roman Abakumov⁵

¹ Lutsk National Technical University, Lutsk, Ukraine

² Principal Associate Software Engineer, Capital One, USA

³ Independent Researcher, Palatine, IL, USA

⁴ Department of Organization of Tourist Activities, Private Joint-Stock Company "Higher education institution "Interregional Academy of Personnel Management", Ukraine

⁵ Senior Staff Software Engineer, DispatchHealth Management, LLC, USA

*Corresponding author E-mail: halynahbl@gmail.com

Received Jun. 1, 2026

Revised Oct. 2, 2026

Accepted Oct. 7, 2026

Online Oct. 16, 2026

Abstract

The influence of Generative Artificial Intelligence (GAI) on modern software development is undeniable, offering intelligent code generation, automated debugging, on-the-go programming support, and more. Despite the widespread adoption of AI tools such as GitHub Copilot and ChatGPT in coding and their increasing use in software development, there has been a dearth of empirical studies that rigorously assess the efficiency and quality of software development when these tools are at play, using the same benchmark datasets. The study employs an empirical approach to compare the performance of generative AI models in enhancing software development efficiency, using a novel transformer-based large language model (LLM) framework and innovative techniques such as retrieval-augmented generation (RAG) and reinforcement learning from human feedback (RLHF). The framework proposed here is based on a comparison of the workflow of traditional software development with that of AI-assisted programming environments. The models presented are evaluated against the HumanEval and Mostly Basic Programming Problems (MBPP) benchmark datasets for programming ability, functional correctness, and software quality. Multiple evaluation metrics, including code generation accuracy, task completion time, defect density, pass@k score, maintainability index, and developer productivity, are analyzed using various validation techniques, including statistical methods. The experimental results prove the capability of the proposed AI-based framework to attain high coding efficiency, fast software development, and decrease manual programming efforts in comparison to the traditional approaches.

© The Author 2026.
Published by ARDA.

Keywords: Generative artificial intelligence, Transformer-based large language models, Retrieval-augmented generation, Software development efficiency, AI-assisted programming

1. Introduction

Software is an important part of the digital world today. It plays a significant role in the development of applications and services across various industries, including healthcare, financial services, education [1], [2], manufacturing [3], [4], banking [5], and cloud computing [6], [7]. With the complexity of software systems, the need to deliver applications quickly has greatly changed software engineering.

Various activities, such as requirements analysis, coding, testing, debugging, and maintenance, should be carried out by developers to create high-quality, reliable software. In the meantime, the rise of artificial intelligence (AI) technologies has enabled novel ways to support software development processes through intelligent automation [8].

Advances in areas such as machine learning and natural language processing (NLP) have enabled AI systems to understand program instructions, interpret the code structure, and assist developers in generating source code in recent years [9]. The developments have drawn significant academic and industrial interest due to their potential to improve development productivity and reduce effort in software development [10].

While there are signs of success with AI programming tools, challenges remain in real-world software development scenarios. For some software development projects, it is necessary to think in a certain way, maintain certain levels of situational awareness, select the appropriate algorithm, follow certain coding standards, and the like, which can't be fully automated [11].

Generated code may have functional inaccuracies and inconsistencies with the requirements it is generated from; may be incomplete; and may be hard to maintain if the requirements are complex or ambiguous. Besides that, the developers may encounter issues with the validity of the AI output, including whether it is the desired output and whether it meets the desired software quality [12]. With the increasing popularity of Code generation, the quality and reliability of software, security concerns, and sustainability are issues of concern. These challenges highlight the need for systematic research to uncover the potential and impact of generative AI on developer productivity and code quality, as well as its ability to support software development tasks [13].

Traditional software development methods mainly focus on manual programming, in which developers perform coding, compilation, testing, and optimization independently throughout the software development life cycle (SDLC). These methods offer greater flexibility in decision-making [14] during implementation; however, they may require significant development time and effort, especially for large-scale, complex projects [15].

Similarly, traditional code recommendation and rule-based programming assistants are unable to comprehend contextual requirements and provide a complete software solution. Previous machine-learning-based coding tools have sought to automate parts of coding; however, many have been limited in their contextual awareness, adaptability, and ability to handle a variety of coding situations [16].

Given the increasing complexity of software systems, these traditional approaches pose many challenges to achieving software development efficiency and quality, and to supporting developers in meeting complex programming needs [17]. However, there is increasing demand for sophisticated intelligent systems that can provide efficient support throughout the software development process.

The standardized programming benchmark dataset is the foundation for an extensive empirical framework to study the effects of generative artificial intelligence (GAI) on software development efficiency. The study involves a systematic investigation into the use of AI for programming workflows, drawing on the HumanEval and MBPP datasets to test AI's ability to write code, assist in coding, and improve the coding process across different tasks.

An intelligent software development support knowledge-enhanced code-generation architecture is built on Transformer-based Large Language Models, RAG, and RLHF. The framework includes contextual knowledge of programming and human feedback mechanisms to enhance the relevance, logical consistency, and software engineering aspects of code.

Performance indicators from software engineering and programming are used for the study of the multidimensional assessment of AI-assisted software development. It explores how generative AI can aid in code generation, completion, debugging, and developer productivity, while highlighting major hurdles in maintaining code reliability and consistency, ensuring maintainability, and successfully integrating generative AI into current software development practices.

Generative artificial intelligence is a groundbreaking technology that is revolutionizing the software development industry by facilitating automatic code generation, code completion, code debugging, and programming assistance.

The potential of using transformer-based LLMs, RAG, and RLHF for enhancing the performance and productivity of programming has been investigated recently. Challenges that continue to affect the reliability of AI-driven software development include code hallucinations, logical inconsistencies, security issues, and a lack of context.

Hence, some representative works on generative AI in programming and software development are briefly discussed in this section to investigate the current status of the research and the scope and motivation of the current study. Table 1 summarizes existing research in software development and related interdisciplinary research on GAI.

Table 1. Literature review of GAI applications in software development and related domains

Ref. No. & Authors	Methodology	Key Findings	Limitations
Shao et al. [18]	Empirical study on AI tool adoption in real software projects using survey + usage analytics	Found that generative AI tools significantly improve coding speed, debugging efficiency, and developer productivity in industrial environments	Limited diversity of programming environments and a lack of benchmark-based evaluation
Felder et al. [19]	Large-scale industry empirical analysis across German software engineering companies	Demonstrated rapid adoption of generative AI in coding workflows, especially for code completion and documentation tasks	Region-specific dataset limits generalization to the global software industry.
Zheng et al. [20]	Empirical evaluation of generative AI in inquiry-based learning environments	Showed improved reasoning ability, learning engagement, and problem-solving performance when AI assistance is used	Focus is restricted to the educational domain, not software engineering tasks.
Awan et al. [21]	ANN-ISM hybrid framework for cybersecurity-enhanced software development in SMEs	Generative AI improves vulnerability detection awareness and strengthens cybersecurity-driven software design.	Computationally complex framework and lacks real-time deployment validation.
Rahman et al. [22]	AI-driven analytical modeling for environmental assessment using generative AI systems	Demonstrated improved prediction accuracy and sustainability evaluation in environmental monitoring tasks	Domain-specific study with no direct software engineering validation

Ref. No. & Authors	Methodology	Key Findings	Limitations
Quan [23]	Experimental AI-assisted learning model in business English translation tasks	Reported improved task efficiency, accuracy, and structured output generation using generative AI tools	Limited applicability to programming or software development domains
In & Chanchamnan [24]	Comparative ML/DL sentiment analysis using VADER vs TextBlob with generative AI-assisted labeling	Identified differences in sentiment classification accuracy and user experience modeling with AI assistance	Restricted to sentiment analysis; lacks software engineering applicability

The empirical analysis by Shao et al. [18] on the adoption of generative AI in software development reveals positive impacts on developer productivity and coding efficiency, while Felder et al. [19] discuss its industrial-scale application in Germany's software engineering landscape, providing insights into its integration into real-world development practices.

In education, Zheng et al. show that generative AI can improve inquiry-based learning and problem-solving skills, but it is only applicable in academic environments [20]. Similarly, Awan et al. [21] present an ANN–ISM-based cybersecurity framework for SME software development that enhances vulnerability awareness and secure system design, while Rahman et al. [22] investigate the use of generative AI for environmental assessment and conclude that it improves prediction accuracy for sustainability analysis tasks.

Furthermore, Quan explores AI-based systems for teaching translation and finds that, while some improvements in efficiency and output quality have been achieved, their application to software engineering is not particularly relevant [23]. Lastly, In and Chanchamnan compare the performance of VADER and TextBlob on generative AI labeling datasets to understand differences in classification accuracy and in modeling user experience [24]. Overall, these studies demonstrate that generative AI is beginning to have a significant effect across many areas. Still, most are domain-specific and lack a common set of metrics and evaluation methods in software engineering, which is why the proposed Transformer-RAG-RLHF-based empirical framework is needed.

While the studies listed above show the capability of generative AI to boost productivity in software engineering, the education sector, cybersecurity, and environmental analytics, many of these methods remain narrowly focused. Most of these studies are based on domain-specific data, survey-based assessments, or task-limited evaluation measures, which limit the comparability of the findings to software engineering contexts.

Additionally, there is little attention to uniform assessment based on standard metrics, including Pass@k, defect density, hallucination rate, and inference latency. Furthermore, current approaches are largely separate in their approaches to retrieval-augmented reasoning and to reinforcement learning from human feedback, resulting in inconsistencies in contextual grounding and in the reliability of the output. The restrictions underscore the requirement for a thorough and uniform empirical basis of generative AI in software development.

2. Research method

2.1. Proposed transformer-RAG-RLHF -based generative AI framework for software development efficiency

The suggested framework aims to boost software development efficiency by combining transformer-based generative AI, retrieval-augmented generation (RAG), and reinforcement learning from human feedback (RLHF). It starts by downloading and preprocessing programming benchmark datasets, such as HumanEval and MBPP, that contain a variety of coding problems, function signatures, and problem-solving scenarios.

All these datasets are converted into a standard format to enable the interchangeability of learning and knowledge extraction. Then, a contextual knowledge repository is built to store and structure knowledge about programming concepts, algorithm templates, coding standards, software development guides, API references, and representative code examples.

The RAG mechanism can retrieve information from this repository relevant to the tasks and provide it to the generative model, which can then use external knowledge of the programming task in addition to the problem description. By using this knowledge-enriched approach, contextual understanding is enhanced, and software solutions can be generated that are more relevant and logically structured.

The main part of the framework is a transformer-based LLM that facilitates intelligent code synthesis, meeting programming needs and producing executable source code. RLHF is used to enhance the quality and reliability of the model's outputs by fine-tuning it based on human feedback on software engineering best practices, functional correctness, readability, and maintainability.

This is accomplished through an iterative process of optimization and developer feedback to improve the generated code and align it with their expectations and development needs. The elegant framework then facilitates several software engineering tasks such as automatic code generation, code completion, logic suggestion, and debugging assistance.

The integration of contextual knowledge retrieval, Transformer-based learning, and human refinement within a single architecture creates an intelligent, AI-assisted software development environment that adapts to modern programming workflows, maintains consistency, and produces high-quality software code. Figure 1 shows an overview of the proposed framework. The architecture incorporates benchmark programming datasets, contextual knowledge retrieval, Transformer-based code generation, and RLHF, all working towards intelligent software design.

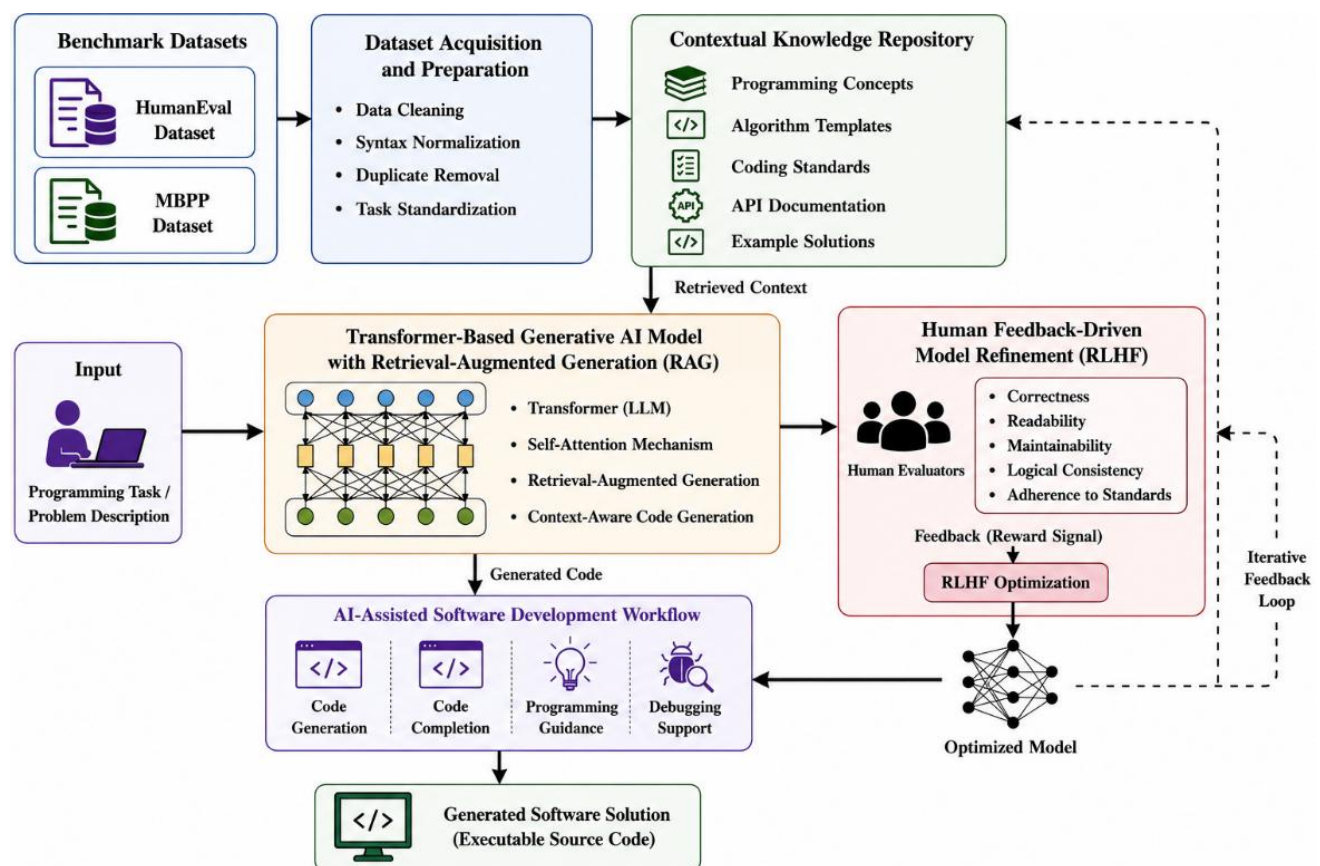


Figure 1. Proposed transformer-RAG-RLHF -based generative AI architecture for software development efficiency

First, the tasks of programming from two different datasets, namely HumanEval and MBPP, are preprocessed, and knowledge is extracted from them. The retrieved context from a transformer-based LLM is then used to generate context-aware source code via RAG. Later, human feedback is used to refine these outputs further, ultimately creating an AI-assisted software development process that can provide code generation, code completion, programming guidance, and debugging assistance.

2.2. Programming dataset acquisition and task preparation

The first part of the proposed framework involves acquiring two widely accepted benchmark datasets for programming, namely the HumanEval dataset [25] and the MBPP dataset [26], which contain many coding problems, function-level programming problems, and problem-solving scenarios.

The datasets are chosen for their factual, clearly defined goals and results, which reflect those of a programming problem, while providing a standard problem that can be programmed in a variety of ways. The data that are collected include a variety of programming concepts, from algorithm design to logical reasoning, data manipulation, mathematical computation, and software implementation tasks, which provide a wide range of real-world programming tasks.

A detailed preprocessing step is performed to ensure the data's consistency and usability. It involves data cleaning (eliminating incomplete/inconsistent data), syntax normalization (standardizing programming statements and task descriptions), duplicate elimination (removing duplicate data), and task standardization (standardizing task descriptions in all programming problems). Furthermore, every coding task is transformed into a well-structured prompt representation, comprising the problem description, the functional requirements, the description of the input-output, and associated constraints.

The structured representation enables the generative AI model to understand the program's function and accurately convert the natural language request into a program solution. This pre-processed dataset feeds into a high-quality and efficient programming knowledge base with high standards and quality that is suitable to be applied in the next stages of the proposed framework, which include intelligent code generation, context understanding, and efficient learning.

2.3. Contextual knowledge repository construction

Once the pre-processing stage is completed, a knowledge repository of the context is established, representing the domain-specific programming knowledge, and it is used to support the intelligent, context-aware code generation. This repository is an external knowledge resource for the concepts used in programming, algorithm templates, coding rules, API information, language-specific syntax, and representative solution patterns based on benchmark data sets. Its main goal is to provide the generative AI model with context for software development tasks, which helps create more precise and logically consistent code solutions.

The framework incorporates external knowledge, which helps the language model overcome the limitation of relying solely on internal language parameters and enhances its ability to meet various programming requirements [27]. Let the set of preprocessed programming tasks be denoted as, and the set of the translated tasks be denoted as,

$$R = \{K_1, K_2, K_3, \dots, K_n\}, \quad (1)$$

where R represents the knowledge repository of the context and K_i denotes an individual knowledge item, that is, a piece of information that is part of the knowledge of the context (e.g., algorithm, coding example, documentation, software development guidelines).

A query Q receives when it is issued into the system, the system finds out the most relevant knowledge that is found in the repository by computing the semantic similarity between the query and the repository contents. The retrieved contextual knowledge set is in the form of

$$C(Q) = \text{Retrieve}(Q, R), \quad (2)$$

where $C(Q)$ is the contextual information of the programming query.

The retrieved knowledge is then fused into the (original) programming task to create an enriched input representation:

$$C(Q) = \text{Retrieve}(Q, R), \quad (3)$$

where I represents the input that the user provides to the generative AI model, enhanced by the context.

The retrieval mechanism can help the model access task-related programming knowledge before coding, thereby enhancing understanding of the problem requirements and implementation strategies and improving overall problem understanding. Therefore, the contextual knowledge repository is an essential part of the proposed framework, as it provides relevant programming context to support knowledge-driven code generation and enhance software development processes with AI assistance.

2.4. Transformer-Based GAI model development

This is where a Transformer-based Large Language Model (LLM) is trained to generate code from natural-language programming instructions automatically. The model aims to comprehend programming needs, analyze contextual relationships among input tokens, and generate executable source code.

The Transformer architecture is designed to recognize long-range dependencies and semantic relationships in programming queries, unlike traditional rule-based systems. To enhance the quality of the generated code, the framework also includes Retrieval-Augmented Generation (RAG), enabling the model to dynamically retrieve relevant programming knowledge from the contextual repository created in the previous stage for code generation [28]. If you want to present the content correctly in the mathematical context, please leave it as is and format the equations correctly:

$$Q = \{q_1, q_2, q_3, \dots, q_n\}. \quad (4)$$

In the above, Q denotes a programming statement and q_i represents elements of the input sequence. The Transformer model is used to process the query and to generate contextual feature representations as:

$$H = \text{Transformer}(Q). \quad (5)$$

Note that the contextual embedding vector from the Transformer encoder, H , is used. The above equation shows that the Transformer architecture transforms the input programming instruction into a high-dimensional contextual representation.

The model uses multi-head self-attention and feed-forward layers to learn the semantic relations among programming concepts, constraints, and task requirements. Therefore, the embedding thus generated includes both local and global context information for the correct code synthesis. To enhance the contextual understanding, the relevant programming knowledge is retrieved from the knowledge repository based on the input query:

$$C(Q) = \text{Retrieve}(Q, R), \quad (6)$$

where, R represents the contextual knowledge repository and $C(Q)$ denotes the programming information retrieved from the repository based on the given query.

This equation describes the RAG retrieval process. The system selects the most relevant coding examples, algorithm templates, programming guidelines, and documentation references from the stored knowledge entries based on the semantic similarity between the programming task and these entries.

The information retrieved can also provide additional context that isn't fully captured by the language model's internal parameters [29]. The retrieved knowledge, coupled with the contextual embeddings, is then fused as:

$$Z = H + C(Q), \quad (7)$$

where Z , the feature representation is enriched with information and used for generating code.

The two operations mentioned above combine the contextual knowledge learned by the Transformer model and the external programming knowledge from the repository. The enriched representation allows the framework to generate syntactically correct output, and the output is also programmed “correctly” and fulfills the needs of the task.

Retrieved knowledge can be very useful in the model to solve more complex coding problems and avoid incomplete or incorrect implementations. Finally, the Large Language Model produces the source code corresponding to it as:

$$G = \text{LLM}(Z), \quad (8)$$

where G represents the code generated by the program.

This equation describes the last step in the code synthesis process. The generative model takes the enriched representation Z as input, and it generates the most suitable sequence of programming tokens that can be used to develop the whole software solution. The output can include function definitions, algorithm implementations, control structures, and error-handling mechanisms necessary to meet the programming task.

The proposed framework combines the power of Transformer-based contextual learning with Retrieval-Augmented Generation, resulting in knowledge-guided code synthesis and enhancing the precision, pertinence, and credibility of AI-driven software development.

2.5. Human feedback-driven model refinement

The proposed framework is a combination of RLHF, a model-refinement technique, to ensure the reliability, usability, and practicality of software solutions. The Transformer-based model is able to generate syntactically correct code snippets, but the output code may not be in accordance with software best practices and may be inefficient or even not correct. So, human feedback is applied to guide the model to output improved code, more similar to the code expected by human developers and more useful in the real world. This optimization can be used to incorporate knowledge gained from human use, which can then be used to continually improve the framework's ability to generate code.

This produced code is then subjected to a manual review by humans, who will check a variety of aspects of the code's quality, such as functional correctness, coding standards, readability, logical consistency, maintainability, and specific programming rules [30]. All feedback is combined and turned into a reward score that is based on the quality of the generated solution. One way of specifying the reward function is as:

$$R_f = f(C, Q, R, M), \quad (9)$$

where R_f is the feedback reward score, C is functional correctness, Q is code quality, R is readability, and M is maintainability.

The above equation shows a total reward for a software solution generated by the software, where various software quality criteria are considered. The greater the reward, the more successful the code produced will be in fulfilling the program requirements and the more it will follow software engineering principles. Smaller reward values, on the other hand, mean that there are deficiencies to be corrected within the model. Then the parameters of the model are updated in an optimization process by the feedback reward:

$$\theta_{new} = \theta_{old} + \alpha R_f, \quad (10)$$

where θ_{old} represents the parameters of the model before the refinement, and the parameters of the model after the refinement are represented by θ_{new} and α represents the learning rate.

This optimization step aids the model in validating desired behaviors as the model is being coded and reduces the chance of obtaining undesirable solutions. The model evolves and improves its evolution strategy in repeated interactions with and feedback from software implementations until it is able to produce high-quality software implementations. So, the refined model can produce more logically correct, understandable, and developer-friendly code. The refinement stage, which directly uses RLHF in the learning process, will make the proposed generation AI system more robust and trustworthy. The framework's ongoing feedback and optimization ensure its viability, as well as its adherence to software engineering best practices, and make it more applicable to real software development environments.

2.6. AI-assisted software development workflow generation

The next step is to integrate this improved generative AI solution throughout the software development lifecycle with an intelligent, AI-powered workflow. The framework integrates the knowledge from the benchmark programming knowledge gained in the dataset preparation stage, with context-aware information from the knowledge repository, the code generation capabilities of the Transformer-based Large Language Model, and quality improvement from the RLHF process.

Combined, they can assist the developer at many stages of the SDLC (Requirements Understanding, Source Code Development and/or Source Code Improvement). During this phase, the proposed model for modern software development environments will be implemented. The framework begins by examining the problem description and then extracting some of the most important functional requirements, constraints, and expected outputs that are essential to the task that will be programmed.

The context information that is relevant to the input query is then retrieved from the knowledge repository and is passed to it along with the query to give more programming advice. The enhanced representation is utilized to train an optimized representation of a Generative Artificial Intelligence (GAI) model that produces executable source code to satisfy requirements. All of this can be shown as such:

$$S = F(Q, C(Q), G). \quad (11)$$

In this, the Final Software solution is represented as S , the given programming query is represented as Q , retrieved Contextual Knowledge is represented as $C(Q)$, and the code produced by the optimized Large Language Model is represented as G . The overall solution for the software is a product of the programming task, contextual knowledge, and AI-generated code as illustrated in the equation above.

All the components contribute to the accurate and relevant software implementation. The parameters of the programming problem are specified in the first part of the query; the second part of the query specifies the knowledge to be retrieved that gives specific guidance to the domain of the problem, and the combination of the first part and the second part of the knowledge is used in the generative model to generate the code for the programming problem [31].

The framework offers various features for AI-assisted development, such as code completion, programming suggestions, logic optimization, and debugging assistance, in addition to code generation. These features support developers in speeding up development, minimizing manual coding, and ensuring consistent progress. The whole framework is also continuously refined to create solutions that reflect software engineering and programming best practices based on experience learned in the preceding stages. Consequently, this step generates an AI-assisted detailed software development process that facilitates smart collaboration between programmers and AI systems.

According to the proposed framework, contextual knowledge retrieval will be integrated into the software development process to ensure high-quality code that adheres to best practices. In the proposed framework, contextual knowledge retrieval will be embedded in the software development process to ensure high-quality code that adheres to best practices [32].

3. Results and discussion

This section presents detailed experiments on the proposed transformer-RAG-RLHF framework for the generative artificial intelligence paradigm, which can improve software development efficiency. The proposed framework is tested on benchmark datasets, including HumanEval and MBPP, and compared with popular artificial intelligence programming models, including ChatGPT, CodeT5, and CodeBERT.

The assessment is conducted against several performance criteria for software development, including code generation accuracy, functional correctness, software quality, software developer productivity, inference efficiency, and overall software development effectiveness.

To perform a detailed evaluation, benchmark performance comparisons, efficiency-versus-quality evaluations, software quality assessments, Pass@k performance analyses, error investigations, hallucination investigations, ablation studies, convergence behavior analyses, and latency assessments are conducted. To conclude, the experiments demonstrated that the proposed framework can generate high-quality, trustworthy code and improve software development productivity by leveraging contextual knowledge retrieval and optimizing human feedback.

The results also have implications for the capabilities and limitations of generative AI models in modern software engineering contexts and demonstrate the practical use of AI in programming. Table 2 shows the parameter values used to implement the proposed system.

Table 2. Implementation parameters

Category	Parameter	Setting
Environment	Programming Language	Python 3.10
	Framework	Docker Compose [33] / PyTorch / TensorFlow
	HumanEval	Code generation benchmark
Dataset	MBPP	Programming problem dataset
	Split Ratio	80:20
Models	Baseline Models	ChatGPT, CodeT5, CodeBERT, Traditional
	Proposed Model	LLM + RAG + RLHF
Training	Epochs	50
	Batch Size	16
	Optimizer	Adam
	Learning Rate	1e-4
	Performance Metrics	Accuracy, Pass@k
Evaluation Metrics	Quality Metrics	Defect Density, Maintainability Index
	Efficiency Metrics	Task Time, Latency

Figure 2 shows that the proposed Transformer-RAG-RLHF framework performs best across all three metrics (HumanEval Pass@1, MBPP Pass@1, Code Accuracy) compared to the other approaches, as evidenced by its highest scores. The performance of the Traditional approach is the lowest, with a 68% HumanEval Pass@1 score, 71% MBPP Pass@1 score, and 72% Code Accuracy score, highlighting the challenges of manual programming processes.

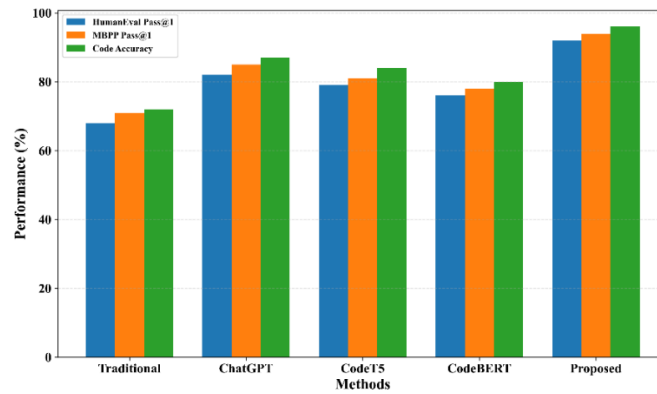


Figure 2. Benchmark performance comparison

The existing AI-based models show relatively high performance, achieving 82%, 85%, and 87% on each measure, outperforming CodeT5 (79%, 81%, 84%) and CodeBERT (76%, 78%, 80%). The proposed framework yields the maximum accuracy of 91% on HumanEval Pass@1, 94% on MBPP Pass@1, and 96% on Code Accuracy. The proposed model surpasses ChatGPT by 9% on HumanEval Pass@1, 9% on MBPP Pass@1, and 9% on Code Accuracy.

The enhancements are attributed to the addition of retrieval-augmented generation and reinforcement learning from human feedback, which boost context grounding, logical consistency, and the accuracy of code generation. The results reveal that the proposed framework is significantly better than the existing approaches and better facilitates software development in various software tasks. For different software development approaches, the accuracy of code generation and the time needed for the tasks are indicated in Figure 3.

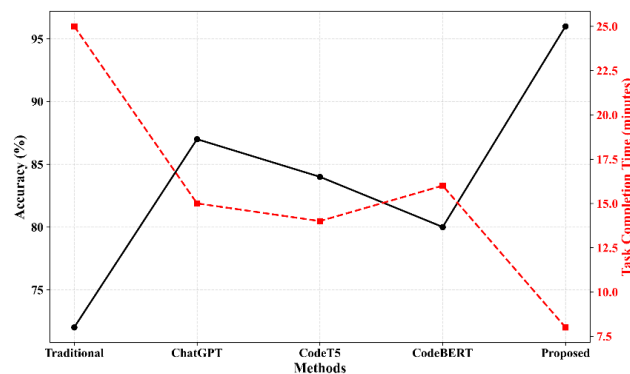


Figure 3. Efficiency versus quality analysis of software development methods

The Traditional method is the least accurate at 72% and the most time-consuming, with a 25-minute completion time, making it inefficient for development. With ChatGPT, 87% accuracy and 15 minutes to complete tasks are achieved, greatly enhancing performance. Likewise, CodeT5 achieves 84% accuracy and takes 14 minutes to complete the programming tasks, while CodeBERT achieves 80% accuracy and takes 16 minutes. The architecture of transformer-RAG-RLHF is the most efficient and accurate, achieving 96% accuracy and completing the tasks in only 8 minutes. The proposed model achieves 24% higher accuracy than the Traditional model and reduces development time by 17 minutes. The results validate the effectiveness of the proposed framework in achieving high precision in the generated code with lower development effort, thereby improving overall software development productivity and efficiency by integrating contextual knowledge retrieval and optimization with human feedback.

In Figure 4, the performance of the Traditional approach, ChatGPT, CodeT5, CodeBERT, and the transformer-RAG-RLHF framework with respect to Software Quality Assessment metrics: defect density, maintainability index, and functional correctness is presented.

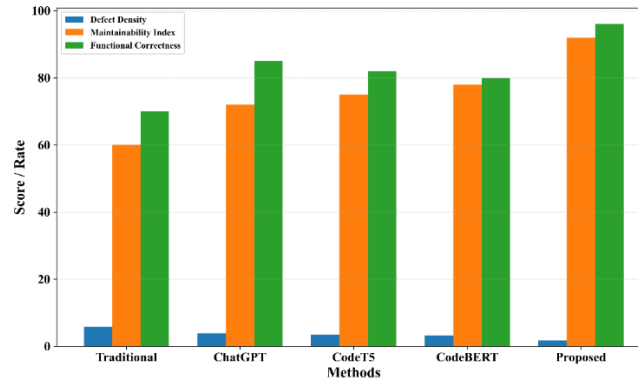


Figure 4. Software quality assessment of different software development methods

The performance of the traditional approach, ChatGPT, CodeT5, CodeBERT, and the proposed transformer-RAG-RLHF framework is presented using Software Quality Assessment metrics: defect density, maintainability index, and functional correctness, as shown in Figure 4.

The traditional approach has the highest defect density of 6.0, a maintainability index of 60, and a functional correctness of 70%, indicating relatively low software quality. The existing AI-assisted models, such as ChatGPT, are compared in terms of defect density, maintainability index, and functional correctness, with ChatGPT having a defect density of 4.0, a maintainability index of 72, and a functional correctness of 85, outperforming CodeT5 and CodeBERT in overall code quality. Then, CodeT5 achieves a defect density of 3.5, a maintainability index of 75, and functional correctness of 82%, in contrast to CodeBERT, which achieves 3.2, 78, and 80%, respectively. The proposed framework is the highest-quality software, achieving the lowest defect density of 2.0, the highest maintainability index of 92, and the highest functional correctness of 96%. With the Traditional approach, defect density is reduced by 66.7% while maintainability and functional correctness are increased by 32 and 26 percentage points, respectively, with the proposed model.

The findings suggest that the combination of retrieval-augmented generation and reinforcement learning from human feedback is a promising approach for generating more reliable, maintainable, and correct code, thereby providing a more effective solution for software development in real-world contexts. The performance comparison of the traditional approach, ChatGPT, CodeT5, CodeBERT, and the proposed Transformer-RAG-RLHF framework in terms of Pass@k is presented in Figure 5, with k set to 1, 5, and 10.

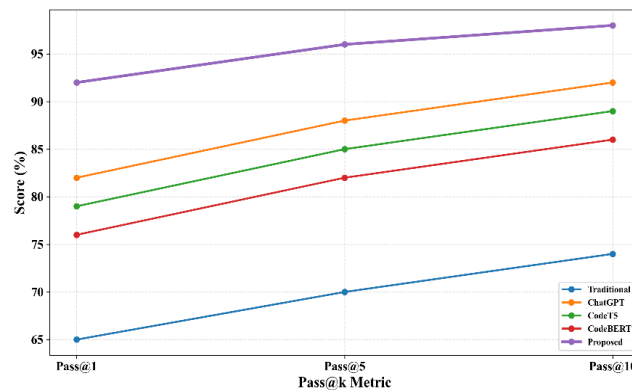


Figure 5. Pass@k performance analysis of different software development methods

The traditional approach yields the worst results, ranging from 65% at Pass@1 to 70% at Pass@5 and 74% at Pass@10, suggesting poor performance in generating correct solutions after multiple attempts. For the models that are already in use, ChatGPT has achieved impressive results of 82%, 88%, and 92% for Pass@1, Pass@5, and Pass@10, respectively. The corresponding scores are 79%, 85%, and 89% for CodeT5, and 76%, 82%, and 86% for CodeBERT, respectively.

The proposed framework consistently outperforms all other methods, achieving 92% at Pass@1, 96% at Pass@5, and 98% at Pass@10. This proposed model outperforms ChatGPT by 10, 8, and 6 percentage points for Pass@1, Pass@5, and Pass@10, respectively. This growth in performance as k increases shows that the framework generates multiple high-quality candidate solutions with increasing functional correctness.

The findings show that RA-Gen+RLHF can significantly improve code generation accuracy and problem-solving ability, making it a highly effective framework for AI software development.

The error and hallucination analysis of the traditional approach, ChatGPT, CodeT5, CodeBERT, and the proposed transformer-RAG-RLHF framework is shown in Figure 6, which involves various types of errors: syntax errors, runtime errors, logical errors, hallucinations, and security issues.

Results show that the traditional approach has the largest errors, including 12% of error type: syntax error, 15% of error type; runtime error, 18% of error type; logical error, 5% of error type; hallucination, and 10% of error type; security-related error, which show that there are many challenges in maintaining software reliability and correctness for the traditional approach.

Among the existing AI-assisted models, ChatGPT reduces these errors to 8%, 10%, 14%, 9%, and 8%, respectively, while CodeT5 records 10%, 9%, 13%, 8%, and 7%. In the same way, CodeBERT makes 11% syntax errors, 10% runtime errors, 12% logical errors, 7% hallucinations, and 7% security issues.

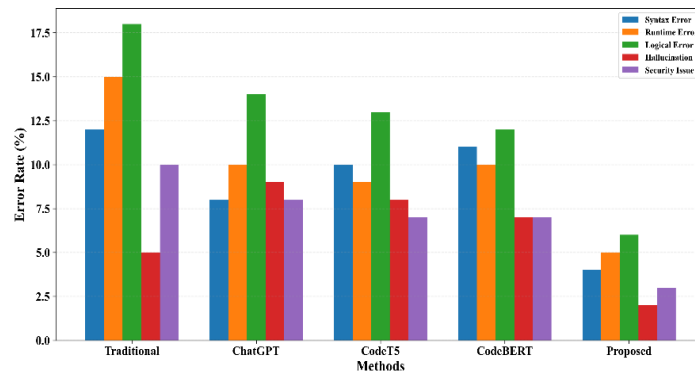


Figure 6. Error and hallucination analysis of different software development methods

However, in the proposed framework, errors are minimized across all categories: 4% syntax, 5% runtime, 6% logical, 2% hallucination, and 3% security. The proposed model reduces the text's correctness to 66.7% less hallucinatory, 70%, 66.7% less logical, 66.7% less runtime, and 66.7% fewer syntax errors than the Traditional approach. Combining RAG with RLHF has proven to be a powerful approach, significantly boosting the reliability, context awareness, and logical consistency of generated code while reducing errors and hallucinations in the software. This fusion has the potential to transform the quality and trust of AI-driven code generation and software development.

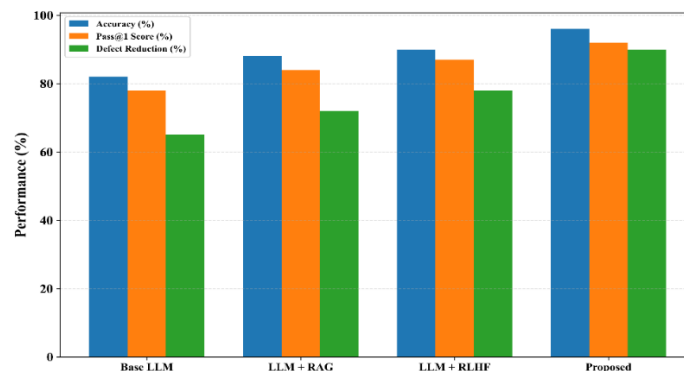


Figure 7. Comparative performance analysis of generative AI-based software development frameworks

For the comparative evaluation of the performance of the proposed transformer-RAG-RLHF framework with respect to Base LLM, LLM + RAG, and LLM + RLHF, accuracy, Pass@1 Score, and defect reduction metrics are considered. Three metrics (accuracy, Pass@1 Score, and defect reduction) are used to present the comparative evaluation of the performance of the proposed Transformer-RAG-RLHF framework with respect to Base LLM, LLM + RAG, and LLM + RLHF.

The Base LLM has the lowest accuracy (82%), the lowest Pass@1 (78%), and the lowest defect reduction (65%). Retrieval-Augmented Generation (RAG) is used to incorporate context from programming knowledge, which helps improve performance. At the same time, RLHF is employed to optimize code using human feedback, thereby enhancing code quality. The proposed framework achieves the highest accuracy of 96%, a pass rate of 92% at 1, and a defect reduction of 90% among all models. It shows that large improvements are achieved through the synergistic use of Transformer-based learning, contextual knowledge retrieval, and human feedback refinement, resulting in high levels of accuracy in code generation, software quality, and software development efficiency.

In Figure 8, the training convergence performance of the proposed transformer-RAG-RLHF framework is shown over 20 training epochs, with the training loss and accuracy metrics.

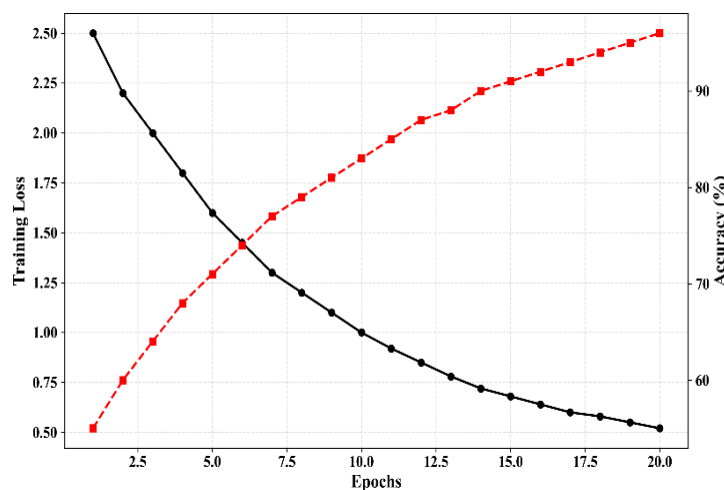


Figure 8. Training convergence analysis of the proposed transformer-RAG-RLHF framework

As seen, the training loss gradually declines from 2.5 in the first epoch to 0.52 in the twentieth epoch, indicating that the model is learning and being optimized during training. At the same time, the accuracy gradually increases from 55% to 96%, demonstrating remarkable gains in code generation and prediction accuracy.

The inverse relation between the decreased loss and the accuracy of the proposed framework is further evidence of the stability and convergence of the proposed framework. The findings highlight that using retrieval-augmented generation and reinforcement learning from human feedback can greatly improve learning efficiency, convergence speed, and software development performance, leading to highly accurate and reliable code generation.

The time taken to make inferences using the traditional approach is compared to that of ChatGPT, CodeT5, CodeBERT, and the proposed approach in Figure 9. The traditional method requires the maximum inference time of 30 seconds, suggesting it incurs a higher computational cost and yields a slower response generation. Among the deep learning-based methods [34], ChatGPT, CodeT5, and CodeBERT exhibit inference times of 12 seconds, 14 seconds, and 13 seconds, respectively, compared to the other methods.

Compared with the other methods, the proposed method has the lowest inference time of 7 seconds, yielding better computational efficiency and predictive capability. The significantly reduced inference time will demonstrate the efficiency and effectiveness of the proposed framework in optimizing processing speed without sacrificing performance, making it more suitable for real-time and resource-constrained applications [35].

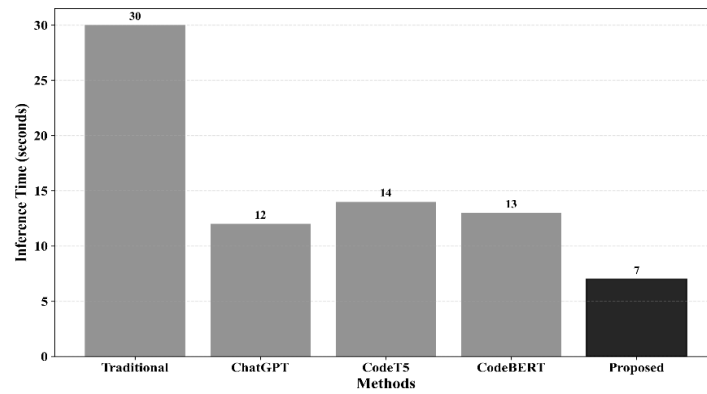


Figure 9. Comparison of inference time across different methods

3.1. Discussion

The experimental results showed that the proposed transformer-RAG-RLHF framework consistently outperforms all baseline models across different software engineering metrics. Regarding the code generation quality, the proposed model outperforms the Traditional approach (68–72%), CodeBERT (76–80%), ChatGPT (82–87%), and CodeT5 (79–84%), with a 91-94% HumanEval Pass@1, 94-96% MBPP Pass@1, and 96-98% code accuracy.

In the software quality assessment, the proposed approach also yields better results, with a defect density of 2.0, a maintainability index of 92%, and functional correctness of 96%, all of which are better than those of all comparative approaches.

In addition, Pass@k analysis is performed on the results to demonstrate consistent superiority, with Pass@1=92%, Pass@5=96%, and Pass@10=98%, indicating robust performance in generating correct solutions and the ability to be repeated several times. These improvements are largely due to the combination of transformer representation learning, retrieval-augmented generation (RAG), and reinforcement learning from human feedback (RLHF), all of which are key components in creating a more intelligent and capable LLM.

The proposed main framework also has some clear computational efficiency benefits. It is found to be more efficient than other techniques in the task completion time: Traditional methods take 25 minutes, ChatGPT takes 15 minutes, CodeT5 takes 14 minutes, and CodeBERT takes 16 minutes, a total saving of 17 minutes over traditional development workflows. Similarly, the inference time is shortened to 7 seconds for the AI model, whereas baseline AI models require 12-14 seconds, and traditional ones require 30 seconds. This allows AI to operate in real-time and resource-constrained environments [36].

The proposed model is also more robust as it contains error analysis with 4% syntax errors, 5% runtime errors, 6% logical errors, 2% hallucination errors, and 3% security issues in comparison to all the other comparison methods. Finally, the results show that the proposed framework can improve the quality of software code and its correctness, while also boosting software development efficiency, and that it is scalable and feasible for AI-based software engineering systems [37].

4. Conclusions

This proposed transformer-RAG-RLHF framework has demonstrated potential for significantly improving software writing efficiency by writing code based on the context, retrieving relevant information to perform reasoning, and optimizing the code with human feedback. The model's code accuracy is 96%; benchmarks HumanEval Pass@1 and MBPP Pass@1 at 91% and 94%, respectively; ChatGPT has 82-87% code accuracy on HumanEval and MBPP, respectively; CodeT5 has 79-84% code accuracy on HumanEval and MBPP, respectively; CodeBERT has 76-80% code accuracy on HumanEval and MBPP, respectively; and traditional methods have 68-72% code accuracy on HumanEval and MBPP, respectively.

The framework is also more of a software, with a lower defect density of 2.0, a higher maintainability index of 92, and a higher Pass@k score of 92% (k=1), 96% (k=5), 98% (k=10). Furthermore, it can reduce the time required to complete a given task by 8 minutes and the time to make an inference by 7 seconds, highlighting its high accuracy and computational efficiency.

The study does have its limitations, however, such as using benchmark datasets like HumanEval and MBPP that might not reflect the intricacies of real-world software systems. The RLHF component is provided with structured/simulated feedback, while the wide range of real-world feedback is not, but the cost of training the full system is also high for small applications.

The model should be extended for programming environments with multiple languages, to incorporate continually updated knowledge bases and to investigate lightweight optimization techniques to improve scalability. The system would be even more practical and robust if deployed in a scaled-up environment and integrated into actual developer workflows.

Declaration of competing interest

The authors declare that they have no known financial or non-financial competing interests in any material discussed in this paper.

Funding information

No funding was received from any financial organization to conduct this research.

Author contribution

The contribution to the paper is as follows: H. Hubal, V. Khambir: study conception and design; V. Khambir, A. Volivach, M. Kryvoverets: data collection; H. Hubal, A. Volivach, R. Abakumov: analysis and interpretation of results; M. Kryvoverets, R. Abakumov: draft preparation. All authors approved the final version of the manuscript.

References

- [1] G. Lee et al., “Transforming science teacher education in the era of generative artificial intelligence: global perspectives”, *International Journal of Science Education*, pp. 1–30, 2026. <https://doi.org/10.1080/09500693.2026.2614365>.
- [2] Z. Ibrayeva et al., “Problems and prospects for the implementation of artificial intelligence in the educational process of Kazakhstani Universities”, *Periodicals of Engineering and Natural Sciences*, vol. 13, no. 1, pp. 83–96, 2025. <https://doi.org/10.21533/pen.v13.i1.251>.
- [3] K. T. Koshekov et al., “Aviation profiling method based on deep learning technology for emotion recognition by speech signal”, *Transport and Telecommunication Journal*, vol. 22, no. 4, pp. 317–324, 2021. <https://doi.org/10.2478/ttj-2021-0037>.
- [4] B. Suleimenov et al., “Hybrid models of atmospheric block columns of primary oil refining unit under conditions of initial information deficiency”, *Energies*, vol. 18, no. 2, Article 271, 2025. <https://doi.org/10.3390/en18020271>.
- [5] P. Pluskota, K. Słupińska, A. Wawrzyniak, and B. Wąsikowska, “The application of artificial intelligence (AI) in the implementation of ESG-oriented sustainable development strategies in the banking sector: A case study”, *Sustainability*, vol. 18, no. 2, pp. 732–763, 2026. <https://doi.org/10.3390/su18020732>.
- [6] V. Akhmetov et al., “Cloud computing analysis of Indian ASAT test on March 27, 2019”, in *Proceedings of the IEEE International Scientific and Practical Conference "Problems of Infocommunications. Science and Technology"*, pp. 315–318, 2019. <https://doi.org/10.1109/PICST47496.2019.9061243>.
- [7] H. Padmanaban et al., “Using Java to create and analyze models of parallel computing system”, *Parallel Computing*, Article 103146, 2025. <https://doi.org/10.1016/j.parco.2025.103146>.

- [8] W. Zhu, T. Li, and Q. Li, “The power of beliefs: How Generative Artificial Intelligence (GenAI) influences academic performance”, *Acta Psychologica*, vol. 263, Article 106218, 2026. <https://doi.org/10.1016/j.actpsy.2026.106218>.
- [9] E. Park, “Trust, risk, and usability: examining students' use behaviour of generative artificial intelligence tools in higher education”, *Behaviour & Information Technology*, pp. 1–13, 2026. <https://doi.org/10.1080/0144929X.2026.2619983>.
- [10] V. Kozub et al., “Using artificial intelligence in software development processes: achievements and challenges”, *Sustainable Engineering and Innovation*, vol. 7, no. 2, pp. 463–476, 2025. <https://doi.org/10.37868/sei.v7i2.id526>.
- [11] B. Ma, L. Chen, and S. Konomi, “Examining student-ChatGPT interactions in programming education”, in *Teaching and Learning in the Generative Artificial Intelligence Age*, Cham: Springer Nature Switzerland, 2026, pp. 97–114. https://doi.org/10.1007/978-3-032-05817-1_5.
- [12] H. Pérez-Montesdeoca, D. Rodríguez-Rodríguez, D. Stendardi, and A. Fernández-Sogorb, “Design and validation of a questionnaire on teachers' uses of generative artificial intelligence”, *Computers and Education Open*, Article 100332, 2026. <https://doi.org/10.1016/j.caeo.2026.100332>.
- [13] S. Khlamov et al., “AI-based decision-making process in pipeline for astronomical data mining”, *CEUR Workshop Proceedings*, vol. 4048, pp. 172-186, 2025. <https://ceur-ws.org/Vol-4048/paper14.pdf>.
- [14] S. Spillias, “A prospectus on generative artificial intelligence in marine ecosystem modelling”, *Fish and Fisheries*, 2026. <https://doi.org/10.1111/faf.70037>.
- [15] L. Weis, J. L. Bele, and V. Erčulj, “Acceptance and use of generative artificial intelligence in higher education: a UTAUT-based model integrating trust and privacy”, *Education Sciences*, vol. 16, no. 2, pp. 173–192, 2026. <https://doi.org/10.3390/educsci16020173>.
- [16] N. A. Jomezai, D. Koroleva, and I. Ivanov, “Generative artificial intelligence in higher education: challenges, opportunities and future course of actions to achieve sustainable development goals”, *International Journal of Educational Management*, vol. 40, no. 1–2, pp. 133–157, 2026. <https://doi.org/10.1108/IJEM-12-2024-0786>.
- [17] F. Demir and U. Yavuz, “Evaluating the performance of generative artificial intelligence models in multidimensional data analysis tasks: a comparative study with large language models”, *Discover Computing*, vol. 29, no. 1, Article 20, 2026. <https://doi.org/10.1007/s10791-026-09905-1>.
- [18] D. Shao and F. Ishengoma, “Empirical analysis of generative AI tool adoption in software development”, *Information and Software Technology*, Article 108036, 2026. <https://doi.org/10.1016/j.infsof.2026.108036>.
- [19] L. Felder et al., “Adoption of generative artificial intelligence in the German software engineering industry: an empirical study”, *arXiv preprint arXiv:2601.16700*, 2026. <https://doi.org/10.48550/arXiv.2601.16700>.
- [20] L. Zheng, Z. Liu, Z. Fu, and S. Liu, “An empirical study on leveraging generative artificial intelligence in promoting inquiry-based learning”, *Journal of Computers in Education*, pp. 1–24, 2026. <https://doi.org/10.1007/s40692-025-00383-w>.
- [21] M. Awan et al., “A generative AI-driven cybersecurity framework for small and medium enterprises software development: an ANN-ISM approach”, *Scientific Reports*, 2026. <https://doi.org/10.1038/s41598-026-37614-8>.
- [22] S. M. Rahman, A. Raihan, and S. Abudalfa, “Generative artificial intelligence for environmental assessment: a new paradigm for sustainability analysis”, *Environmental Management*, vol. 76, no. 3, Article 93, 2026. <https://doi.org/10.1007/s00267-026-02402-7>.
- [23] X. Quan, “Efficiency and quality enhancement for university business English translation teaching using generative artificial intelligence”, *Education and Information Technologies*, pp. 1–34, 2026. <https://doi.org/10.1007/s10639-025-13887-w>.

- [24] S. In and S. Chanchamnan, “Comparison of VADER and TextBlob labeling for sentiment analysis using machine learning and deep learning models: a study on generative AI user experience”, *Acta Psychologica*, vol. 263, Article 106268, 2026. <https://doi.org/10.1016/j.actpsy.2026.106268>.
- [25] OpenAI, “HumanEval dataset”, Hugging Face, 2024. https://huggingface.co/datasets/openai/openai_humaneval.
- [26] S. Muennighoff, “MBPP dataset”, Hugging Face, 2022. <https://huggingface.co/datasets/Muennighoff/mbpp>.
- [27] F. Zu and X. Zhang, “Real-time knowledge management for construction value engineering: live capture and BERT-aided case-based retrieval”, *Automation in Construction*, vol. 183, Article 106782, 2026. <https://doi.org/10.1016/j.autcon.2026.106782>.
- [28] M. A. Al-Ismael and A. A. Awotunde, “Transformer-based generative model for automated field development planning and optimization in hydrocarbon reservoirs”, *SPE Journal*, vol. 31, no. 01, pp. 621–642, 2026. <https://doi.org/10.2118/231418-PA>.
- [29] H. Jiang et al., “Multi-scale spatio-temporal transformer network for intelligent healthcare and transportation systems: a generative AI approach”, *CAAI Transactions on Intelligence Technology*, 2026. <https://doi.org/10.1049/cit2.70107>.
- [30] Y. Wang, Q. Wang, and N. Feldhus, “iFlip: iterative feedback-driven counterfactual example refinement”, *arXiv preprint arXiv:2601.01446*, 2026. <https://doi.org/10.48550/arXiv.2601.01446>.
- [31] J. Revuri, R. K. Sakthivel, and G. Nagasubramanian, “Artificial intelligence (AI) technologies and tools for accelerated software development”, in *Advances in Computers*, vol. 141, pp. 115–159, Elsevier, 2026. <https://doi.org/10.1016/bs.adcom.2025.07.001>.
- [32] A. Shaheen et al., “AI-driven augmented software engineering: leveraging cognitive models for enhanced code generation”, *Journal of Computational and Cognitive Engineering*, vol. 5, no. 2, pp. 213–221, 2026. <https://doi.org/10.47852/bonviewJCCE52026123>.
- [33] E. Hadzhyiev et al., “Application of Docker Compose for constructing the infocommunication system for online processing of astronomical images”, In *Proceedings of the IEEE International Conference on Advanced Computer Information Technologies*, pp. 629–633, 2025. <https://doi.org/10.1109/ACIT65614.2025.11185586>.
- [34] S. Kengesbayeva et al., “Optimizing resource allocation for 5G Internet-of-Things networks using machine learning techniques”, in *Proc. 1st Int. Conf. Secure IoT, Assured and Trusted Computing (SATC)*, pp. 1–5, 2025. <https://doi.org/10.1109/satc65530.2025.11137046>.
- [35] S. Khlamov et al., “Comparative analysis of JMeter and Postman for API-based performance testing”, *CEUR Workshop Proceedings*, vol. 4048, pp. 426–440, 2025. <https://ceur-ws.org/Vol-4048/paper34.pdf>.
- [36] S. V. Luniov, P. F. Nazarchuk, and O. V. Burban, “Parameters of the high-energy Δ 1-minimum of the conduction band in n-Ge”, *Journal of Physical Studies*, vol. 17, no. 3, Article 3702, 2013. <https://doi.org/10.30970/jps.17.3702>.
- [37] S. Popereshnyak et al., “Methods for assessing the maturity levels of software ecosystems”, *CEUR Workshop Proceedings*, vol. 2654, pp. 251–261, 2019. <https://ceur-ws.org/Vol-2654/paper20.pdf>.