CENG 407

Innovative System Design and Development I

2025-2026 Fall

AI Assisted Programming Platform for Education and Assessment

Literature Review

İrfan Gökdeniz YILMAZ 202111038

Emir Tuğberk TOZLU 202011037

Emir KARA 202011068

Ahmet Kerem ÖZTÜRK 202211405

Salih Cihat YALÇIN 202111034

30.11.2025

Abstract

This literature review provides an analytical overview of the development and use of Large Language Models (LLMs) in various fields. It first introduces the essential principles of Generative AI and the architectural foundations that enable modern LLMs. The study then examines several major LLM systems including ChatGPT, Gemini, Claude, Grok, CoPilot, DeepSeek, and LLaMA highlighting their design goals and distinguishing features. Performance evaluation methods and benchmarking results are also discussed to show how these models differ in efficiency, reasoning capability, and applicability across real-world scenarios.

In the context of education, the review investigates how LLM-based tools are being integrated into teaching and learning practices, emphasizing concrete examples and evidence regarding their influence on student engagement and learning outcomes. Another section focuses on the role of LLMs in computer programming, addressing their usefulness in code generation, debugging assistance, and productivity enhancement, as well as the challenges that arise when incorporating them into software development workflows.

The final part summarizes related technical components such as front-end, back-end, and database systems that support the deployment of LLM-driven applications. Overall, this review aims to provide a clear and comprehensive synthesis of current knowledge on the impact of LLMs in technological, educational, and programming environments.

21.Referances

1.Fundamentals of Generative AI and LLMs

1.1. The Hierarchy of AI

The evolution of modern Artificial Intelligence (AI) can be understood as a layered progression of technology. At the broadest level, AI encompasses all systems designed to replicate human like cognitive functions such as learning and decision making. Within AI, Machine Learning (ML) emerged as a major paradigm by enabling models to learn patterns from data. A significant subset of ML, Deep Learning (DL), relies on multi layered artificial neural networks that emulate the structure of the human brain [1] .These deep neural networks made it possible for models to automatically extract complex features from data, laying the groundwork for generative systems. Generative AI (GenAI) marks the shift from traditional "discriminative" models, which focus on analyzing or classifying existing data, to models that can autonomously create new content text, images, or code that is realistic, coherent, and novel.

1.2. Transformer Architecture

A major breakthrough that accelerated this transformation was the introduction of the Transformer architecture in 2017 [2] . The key innovation behind Transformers is the self attention mechanism, which allows models to evaluate relationships between all words in a sentence simultaneously, rather than processing them sequentially. This enables an understanding of long range dependencies and contextual meaning, similar to how humans emphasize certain words for comprehension. The Transformer architecture became the foundation for influential models such as BERT and the GPT family [3] , enabling them to scale to unprecedented levels and perform a wide range of natural language tasks with remarkable accuracy.

1.3. Pre training, Scaling Las, and Efficiency

Large Language Models (LLMs) are vast probabilistic models trained to predict the next token in a sequence, using enormous amounts of text data. During pre training, these models ingest trillions of tokens from diverse sources like CommonCrawl, Wikipedia, GitHub, and scientific papers. Research such as the LLaMA series demonstrated that high quality data and effective training strategies can allow smaller models to outperform much larger ones [4] , challenging the assumption that parameter count alone determines capability. Scaling laws further show that performance correlates strongly with available compute and data [5] , though efficiency during inference is increasingly important for real world deployment.

1.4. Alignment and Reinforcement

A pre-trained LLM, however, is simply an advanced text generator and is not inherently aligned with human values or intentions. To make models safer, more cooperative, and more reliable, developers apply Reinforcement Learning from Human Feedback (RLHF). This process begins with Supervised Fine-Tuning (SFT), where human-written examples teach the model desirable behavior [6] . Next, a reward model is trained using human-ranked responses to determine which outputs are preferred. Finally, reinforcement learning algorithms such as Proximal Policy Optimization (PPO) adjust the model to maximize these reward signals, making it more helpful, honest, and harmless.

1.5 .Prompt Engineering Techniques

How users interact with LLMs known as prompt engineering also significantly affects performance. Techniques such as zero shot and few shot prompting help guide the model by either relying entirely on prior knowledge [7] or providing examples of the desired output. The chain of thought prompting encourages the model to break down complex reasoning tasks into intermediate steps [8] , greatly improving outcomes when applied explicitly. Role prompting, where the model is assigned to a professional persona or specific identity, helps shape tone, structure, and domain relevance in generated responses.

1.6. Limitations

Despite their power, LLMs come with inherent limitations. They may hallucinate producing confident but factually incorrect statements [9] because they predict text based on statistical patterns rather than grounding their output in verified facts. They can also replicate or amplify biases present in training data, leading to skewed or unfair responses [10] . Additionally, LLMs remain in “black box” systems, making it difficult to fully understand how specific outputs are produced. These challenges highlight the need for ongoing research in transparency, alignment, and responsible deployment of AI systems.

2.LLM Products

The landscape of Large Language Models has evolved into diverse product families, each optimized for specific trade-offs between speed, cost, and reasoning capability. This section outlines the major proprietary and open source models currently defining the state of the art.

2.1. ChatGPT (OpenAI)

OpenAI’s ChatGPT has evolved across multiple generations, culminating in GPT-5. This version introduces a unified architecture with dynamic routing, allowing the model to switch between a fast lightweight mode and a deeper reasoning mode (“GPT-5 Thinking”). The GPT-5 lineup includes variants such as nano, mini, and the flagship model, each balancing cost and capability differently. GPT-5 focuses on advanced reasoning and multimodal understanding but can produce dense, complex text and requires higher computational resources [11] .

2.2. Gemini (Google)

Google’s Gemini 3 family, including variants like Gemini 3 Pro, emphasizes advanced multimodal capabilities across text, audio, images, video, and code, with efficient deployment for complex tasks. Gemini 3 Pro is optimized for high-accuracy reasoning, agentic performance, and long-context processing up to 1 million tokens, while supporting reinforcement learning for multi-step problem-solving. The family prioritizes efficiency through a sparse mixture-of-experts architecture, making it suitable for developer tools, AI agents, and integrated systems, with notable strengths in benchmark-leading output quality and safety [12] .

2.3. Claude (Anthropic)

Anthropic’s Claude models, including Claude Opus 4.1 and Claude Sonnet-4, are built around the principles of Constitutional AI, prioritizing safety, reliability, and structured reasoning. They often generate well-organized and coherent text, though their style can sometimes become overly complex. Claude models are typically favored in scenarios requiring careful, rule-guided behavior and strong alignment with safety constraints [13] .

2.4. Grok (xAI)

xAI’s Grok-4 integrates real-time information from the X platform, allowing it to provide up-to-date responses. It is designed to be conversational, dynamic, and highly responsive to current events. Grok also emphasizes reliability in tasks requiring grounded information, especially in areas like reference retrieval. Its main stylistic characteristic is a tendency toward high-density, complex language.

2.5. Copilot (Microsoft)

Microsoft's Copilot is designed as an assistant integrated across the Windows ecosystem and productivity tools. It excels at general tasks such as summarizing content, drafting text, generating code, and interacting with Microsoft services. However, it requires external validation when used for academic or research heavy tasks due to limitations in handling detailed citation or scholarly material [14] .

2.6. DeepSeek

DeepSeek-v3.1 focuses on cost efficient deployment while maintaining high accuracy in structured tasks like reference generation. It performs well in theoretical and knowledge-based operations and is positioned as a highly competitive low-cost alternative to major proprietary LLMs. Its main limitation is inconsistency in tasks requiring detailed numerical or calculation-heavy reasoning [15] .

2.7 LLaMA (Meta)

Meta's LLaMA family ranging from 7B to 65B parameters is fully open source and trained exclusively on publicly available datasets such as CommonCrawl, GitHub, Wikipedia, and ArXiv. LLaMA emphasizes accessibility, transparency, and customizability. Because of its open-source nature, it has become a widely adopted foundation model for both enterprise solutions and community-built LLM derivatives [16] .

3. LLM Performance Overview

This section provides an integrated evaluation of how modern AI models perform across domains such as medicine, dentistry, supply-chain decision-making, and academic reliability. Overall, frontier models like GPT-5, Gemini 3, Claude 4.x, and Grok-4.1 demonstrate near "expert-level" knowledge, but their strengths and weaknesses vary significantly by task type and domain.

3.1. Performance in Medical & Dental Domains

To evaluate how well the evaluators agreed with each other, Intraclass Correlation Coefficients (ICC) were calculated separately for accuracy and completeness scores. The normality of continuous variables was checked using the Shapiro–Wilk test. Normally distributed variables are presented as mean ± standard deviation (SD), while non-normal variables are presented as median (minimum–maximum).

Based on the normality results, group comparisons were done using ANOVA when there were more than two groups, and the data were normally distributed. When normality was not met, the Kruskal–Wallis test was used. If ANOVA showed significant differences, Bonferroni-adjusted post-hoc tests were applied. If Kruskal–Wallis showed significance, Dunn–Bonferroni post-hoc tests were used. Correlations between scores were examined with Pearson or Spearman correlation coefficients, depending on data distribution. Effect sizes were calculated using eta-squared (η^2) for both parametric and non-parametric tests to assess the practical importance of group differences. All statistical analyses were performed using SPSS for Windows (version 27.0; IBM, Armonk, NY, USA). 5% types I error level was accepted, and statistical significance was set at $p < 0.05$ [17] .

3.2. Supply Chain, Decision Making

This domain evaluates models on progressively harder tasks: theoretical multi-choice items, reasoning-based questions, and open numerical problems.

3.2.1. Numerical vs. Theoretical Tasks

GPT-5 dominates numerical tasks, achieving 92% accuracy when no answer options are provided. By contrast, smaller models like Claude-Haiku 3.5 perform poorly on these tasks (under 20%). For theoretical, single-choice questions, Gemini-3 ranks highest.

3.2.2. Impact of Chain of Thought Prompting

Studies show that implicit CoT (“think step by step”) often fails to improve accuracy and can even decrease performance in complex tasks, particularly for Claude models.
However, explicit CoT, where the model is forced to write out its reasoning, leads to major jumps in accuracy.
DeepSeek, for example, rises from 0% → 80% on medium-difficulty numerical problems when explicit reasoning is required [18] .

3.2.3. Cost Speed Accuracy Balance (AHP Analysis)

When accuracy, latency, and cost are weighted together, GPT-5 mini emerges as the most balanced model—offering high reasoning performance with significantly lower resource usage than the full GPT-5 model.

3.3. Academic Reliability & Hallucination Rates

A key finding of this study is the persistent and structurally embedded problem of hallucination in the bibliographic outputs of most chatbots. Hallucination occurs when a model, unable to retrieve a real source from its training distribution, generates a reference that is grammatically coherent and academically plausible but factually nonexistent. This behavior is not random; it reflects the probabilistic nature of LLMs, which prioritize producing linguistically well-formed output even in the absence of verified factual information. As a result, many fabricated references adopt the appearance of legitimate citations, blending real-sounding author names, common journal titles, and syntactically correct DOIs into entries that can be difficult for non-experts to detect false.

The results of this study show considerable differences across models. While Grok and DeepSeek did not fabricate any references, other systems most notably Copilot, Perplexity, and Claude displayed extremely high hallucination rates, sometimes generating entire lists of references that do not exist in any academic database. A particularly concerning behavior is the production of "structured" hallucinations, in which elements from real works (such as well-known authors or publishers) are mixed with incorrect details like fabricated article titles or invalid publication years. These references appear legitimate at a glance and therefore pose significant risks for students who might accept them uncritically.

Hallucination is also closely tied to the type of document requested. Models were far more likely to fabricate journal articles than books, likely because widely used textbooks appear frequently in their training data, whereas journal content—especially specialized or recent articles is less consistently represented. This helps explain why hallucination was more common in fields dominated by article based scholarship, such as Engineering, Experimental Sciences, and Health Sciences. In some cases, chatbots attempted to satisfy the user's implied preference for recent sources by assigning publication years in the 2020–2025 range to entirely fabricated works, reinforcing the idea that the system's goal is to produce plausible output rather than accurate information.

Overall, hallucination emerges as a fundamental limitation of current LLM based chatbots in academic tasks requiring citation-level precision. Even when models provide real references, they frequently contain subtle inaccuracies that stem from the same generative mechanisms responsible for full fabrication. These patterns underscore the importance of thorough manual verification and highlight the ongoing need for improved training data, better integration with authoritative bibliographic sources, and stronger educational strategies to ensure students use AI tools critically and responsibly [19] .

3.3.1. Fabrication Rates

While Grok and DeepSeek demonstrated high integrity with zero fabricated references, Copilot exhibited significant issues, fabricating all of its journal citations. ChatGPT and Gemini presented a different challenge: they often generated 'hallucinated' citations that appeared plausible by pairing legitimate authors with non-existent article titles [20] .

3.3.2. Document Type Differences

Models are far more reliable with book references (only 12.9% fabricated) compared to journal articles, where fabricated entries jump to 78% [21] . This significant discrepancy highlights that LLMs still struggle with the granularity of academic literature. While books are often treated as distinct, major entities within the model's training data, appearing frequently across multiple sources and editions, journal articles are far more numerous and specific. Consequently, instead of retrieving an exact article, models often resort to a probabilistic "reconstruction" strategy: they hallucinate citations by pairing real, established authors with plausible sounding but non-existent article titles that fit the context. This pattern-matching behavior explains why the error rate for specific journal articles remains critically high, rendering most models unreliable for deep literature reviews without rigorous human verification.

3.4. General NLP & Reasoning Benchmarks

Beyond domain-specific tasks, evaluating LLMs on general Natural Language Processing (NLP) and reasoning benchmarks provides a clearer picture of their fundamental capabilities. While earlier research focused heavily on scaling laws assuming that larger models inherently yield better results recent evidence suggests a paradigm shift. Current benchmarks indicate that architectural efficiency and advanced training methodologies, such as instruction tuning, are often more critical for determinants of performance than raw parameter count alone.

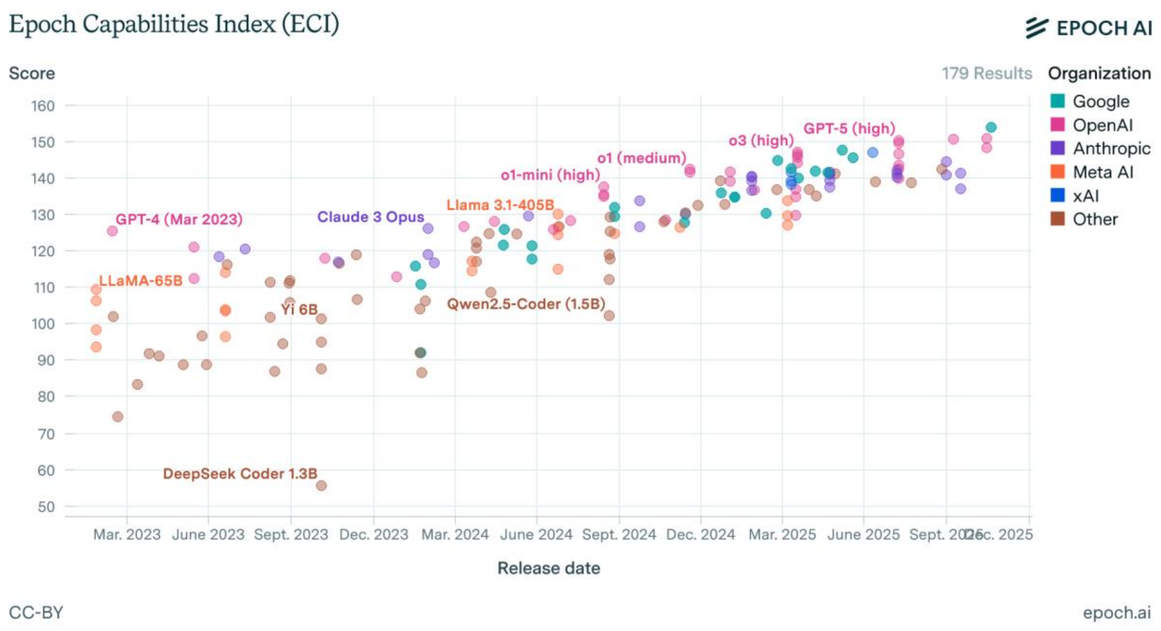

Figure 1: The Epoch Capabilities Index (ECI) showing the progression of general model capabilities over time, highlighting the rapid rise of reasoning-focused models like the GPT-5 and o1 series.

3.4.1. Zero-Shot Reasoning

The results challenge the idea that parameter count alone determines performance. LLaMA-65B outperforms GPT-3 (175B) on major reasoning benchmarks such as PIQA, SIQA, and OpenBookQA. On TriviaQA, it also surpasses Gopher-280B [16] .

3.4.2. Instruction Following & RLHF

InstructGPT (1.3B) is preferred by human evaluators over GPT-3 (175B), demonstrating that RLHF is more influential than model size for alignment and helpfulness. RLHF fine-tuning also. Furthermore, the application of RLHF has been shown to increase model truthfulness by minimizing hallucinations, while simultaneously reducing the generation of toxic outputs by approximately 25%.

These findings show that training strategy and alignment techniques matter as much as, if not more than, raw scale [22] .

4. LLMs in Education

Large Language Models (LLMs) have become one of the fundamental elements of the recent transformation in educational technologies. Leveraging their Transformer-based architectures, these models demonstrate high accuracy in natural language processing, effectively interpret contextual relationships, and generate explanations with clear pedagogical relevance. Unlike traditional knowledge-transfer-based teaching methods, LLM-based systems can instantly adapt to the student's learning pace, needs, and potential misunderstandings. These adaptive features make LLMs stand out as both a dynamic support tool in learning processes and a scalable solution in instructional design.The inclusion of LLMs in educational environments has fundamentally transformed the forms of interaction between students and teaching staff. The capacity to provide pedagogically framed explanations to students' questions and identify conceptual errors transforms these models from simple “response generators” into active teaching components. Various studies in the literature report that the personalized feedback processes offered by LLMs significantly strengthen students' motivation to learn and their self-regulation skills [23], [24].

From the perspective of teachers, LLMs are increasingly being used to alleviate the heavy workload in educational processes. Time-consuming tasks such as preparing course content, simplifying texts, generating exam questions, and conducting rubric-based assessments can be carried out more quickly and consistently thanks to LLMs. Institutional assessment platforms can use these models to identify conceptual errors in student writing and provide structured feedback [25]. These automation processes allow teachers to devote their time to higher-level pedagogical planning.However, the literature also highlights some potential drawbacks of using LLMs. Models sometimes produce context-disconnected or erroneous explanations, which can lead to students acquiring incorrect information. Furthermore, it is stated that excessive use may weaken students' independent problem-solving skills and lead to superficial learning. Indeed, as shown in a study published in the journal Applied Sciences [26], high dependence on LLMs is negatively correlated with students' final performance. Therefore, it is crucial to use LLMs in a balanced and conscious manner from a pedagogical perspective.The integration of LLM-based solutions into education raises many ethical considerations. Training these models on large and heterogeneous datasets naturally increases risks such as privacy concerns, bias generation, and misinformation creation. Clearly defining and institutionally monitoring the processes of collecting, processing, and storing student information are fundamental requirements for ensuring the safe use of these technologies [24].

Findings in the literature reveal that LLM-based systems offer an effective structure that significantly supports teaching processes. Facilitating students' conceptual learning, reducing teachers' operational burden, and providing personalized feedback make these technologies a valuable element for contemporary education models. However, for LLMs to be used efficiently, it is critically important to blend technological capabilities with pedagogical principles and maintain a human-centered approach in the learning process.

4.1. Example Projects and Applications in General Education

LLM-based applications have evolved into a broad ecosystem that offers flexible, scalable, and personalized solutions tailored to different learner profiles in general education. These systems not only transfer knowledge but also provide integrated learning environments that guide, structure, and adapt to the learning process according to individual needs. Along with the digitalization process, the primary goal of LLM-based platforms is to reduce teacher workload, increase learning speed, and create sustainable teaching models suitable for different learning levels. In this context, this section details how institutional, commercial, and academic systems are positioned in educational processes.

4.1.1. Khanmigo (Khan Academy)

Khan Academy's GPT-4-based Khanmigo tool is one of the prominent examples in general education. The system provides an interactive learning process that encourages step-by-step thinking rather than simply providing the solution to the student. When a student takes the wrong approach, the model guides them to the correct conceptual structure with leading questions and hints. This approach not only supports independent reasoning but also reduces cognitive load, thereby contributing to more meaningful and sustained learning [24][27].

4.1.2. Duolingo Max

Duolingo Max represents one of the most successful applications of LLMs in language learning. The platform's AI-powered dialogue system enables learners to interact in real time using natural language. When a student uses an incorrect expression, the model not only provides corrections but also explains the grammatical and semantic aspects of the error in detail, facilitating a deeper learning process. Such explanatory feedback strengthens lasting comprehension in language learning, while the role-playing scenarios offered by the system allow users to experience real communication situations in a safe environment and support the development of conversational fluency [28].

4.1.3. Google LearnLM

Google LearnLM is designed as an extensive system consisting of large language models customized for educational purposes. The model can generate tailored explanations based on the student's age level, learning objectives, and subject area.

Particularly effective in STEM subjects, LearnLM simplifies complex content to an appropriate cognitive level, identifies potential misunderstandings, and suggests learning paths to make the student's progress more efficient. Unlike traditional online content, it demonstrates a more advanced cognitive adaptation capability [29].

4.1.4. OpenStax Tutor

OpenStax Tutor provides a framework that analyzes data from completed activities and identifies individual learning gaps. The system identifies areas where students struggle and creates personalized learning plans. This feature offers a significant advantage, especially in crowded classrooms where teaching tailored to individual needs is not possible. In addition, real-time assessment outputs provide teachers with a strong data foundation for monitoring the overall learning status of the class [30].

4.1.5. Automated Scoring Systems (ETS & Pearson)

Another category frequently used in education is automated assessment and feedback systems. Developed by institutions such as ETS and Pearson, these tools analyze student texts according to rubrics and can make assessments based on criteria such as content structure, logical coherence, and conceptual accuracy. These models reduce teachers' workload by increasing the speed and consistency of assessment, particularly in writing-focused courses and large-scale exams [25]. In addition, the model provides structured feedback on areas for improvement rather than just scoring [31].

4.1.6. PQG Systems (AST + Concept Graphs)

Programming Question Generation (PQG) systems used in programming education leverage Large Language Models (LLMs) to generate coding questions appropriate for the student's level. In such systems, models produce curriculum-aligned content using technical components such as Abstract Syntax Trees (AST) and Local Knowledge Graphs (LKG) [32]. This significantly reduces the time teachers spend preparing new questions while also enabling the generation of more exercises targeting topics that students find challenging.

4.1.7. Multi-agent LLM Systems

Finally, multi-agent LLM systems are more advanced structures based on task sharing among several artificial intelligence agents. In these models, for example, one agent reviews the student's solution and identifies errors, while another agent generates explanatory feedback; yet another updates the learning plan based on the student's progress. Research shows that these multi-agent structures can provide more consistent and reliable educational outputs compared to the responses generated by a single model [33].

The examples at hand demonstrate that LLM-based technologies offer a wide range of functionality, flexibility, and adaptability in educational settings. These applications create scalability in teaching processes while providing students with a more personalized learning experience; they reduce teachers' workload while enhancing pedagogical quality. However, the long-term sustainable evaluation of this technological potential depends on a purpose-driven and controlled integration process that is aligned with pedagogical design principles.

4.2. Effectiveness or Impact of LLMs in Education

Research examining the impact of LLM-based technologies in education focuses on multi-layered areas such as learning outcomes, cognitive processes, motivation, social interaction, assessment accuracy, and contributions to instructional design. Studies conducted in this vein comprehensively reveal both the advantages and limitations of LLM-supported learning experiences.

4.2.1. Learning Performance Impact

Findings related to learning performance indicate that LLM-supported explanations lead to meaningful development, particularly in STEM and programming fields. LLMs can consistently deliver the step-by-step guidance students require when working through complex concepts. Experimental analyses of programming education show that students can distinguish error types more clearly, interpret the logical structure of code more quickly, and experience significant increases in learning speed. Findings from the Intelligent Deep Learning Tutoring System show that LLM-style explanatory feedback enhances students' problem-solving processes and substantially supports conceptual development [26].

4.2.2. Affective (Motivational & Emotional) Effects

The affective dimension is one of the prominent areas of LLM-supported learning studies. Reduced anxiety among students, a more relaxed approach to the trial-and-error process, and a willingness to make mistakes are among the strengths of LLM-based feedback mechanisms. A study has shown that LLM-supported paired programming increases both intrinsic motivation and the perception of benefits in the learning process among students [34]. The non-judgmental nature of LLMs helps students take more risks and view mistakes as a natural part of learning.

4.2.3. Social Interaction & Collaboration Effects

When evaluated in terms of social interaction, LLM-supported collaborative environments have been reported to enhance students' awareness of collaboration. Although the sense of social presence in human-human interaction cannot be fully achieved, some studies show that pair programming with LLM can match human-paired programming in performance outputs and contribute to more systematic error correction [34]. Students can ask more questions and think longer about conceptual issues when working with LLM.

4.2.4. Assessment and Evaluation Impact

Another noteworthy area is the impact of LLM-based systems on assessment and evaluation processes. Assessment models developed by ETS and Pearson can provide highly consistent feedback in terms of content integrity, structure, consistency, and rubric alignment [25]. In large-scale examinations, such systems substantially reduce grading time, lessen the burden on instructors for text-analysis-heavy tasks, and help decrease variability across human evaluators.

4.2.5. Negative Effects & Over-reliance Issues

Applied research also shows that LLM use does not always produce positive results. In a study published in Applied Sciences, Roldán-Álvarez and Mesa (2024) found that over-reliance on LLMs is negatively correlated with student achievement [26]. Based on Spearman correlation results, students who relied heavily on the model demonstrated reduced conceptual depth and a tendency toward more superficial learning. This highlights the need for balanced use in LLM-based learning processes.

4.2.6. Multi-Agent LLM Systems Impact

Recently developed multi-agent LLM systems offer a new approach aimed at improving consistency in learning processes and feedback quality. Having multiple LLM agents independently evaluate the same student solution and generate feedback by taking on different roles reduces error rates and strengthens pedagogical accuracy [33]. Various roles, such as expert, critical, and explanatory agents, support different stages of learning, making the system more robust.

4.2.7. Limitations, Risks, and Ethical Concerns

There are also studies pointing to limitations in the use of LLM. These models may occasionally misinterpret context and generate inaccurate or fabricated information (hallucinations), potentially leading to conceptual misunderstandings through pedagogically inconsistent explanations. Such situations may cause students to believe incorrect information to be true or reinforce their existing misconceptions. Furthermore, concerns related to data privacy, systemic biases, and ethical accountability are critically important, particularly in scenarios involving the storage and processing of student data [24] [33]. The increasing prevalence of LLM-based technologies in educational environments makes it imperative to establish institutional oversight mechanisms to maintain the reliability of these systems.

Overall, LLM-supported solutions offer multidimensional contributions to the educational ecosystem. Increased academic achievement, conceptual clarity, heightened motivation, consistency in assessment processes, and efficiency in instructional design are among the primary contributions. However, considering model errors, the risk of superficiality that excessive use may cause, and ethical issues, LLM integration must be carried out in a controlled, purpose-oriented manner and in line with pedagogical frameworks. When properly structured, these systems offer a powerful learning support mechanism that sustainably improves education quality.

5. Applications of Large Language Models in Computer Programming: A Research-Oriented Survey

Large Language Models (LLMs) have rapidly evolved into essential tools within modern software engineering. Their ability to interpret context, generate code, detect bugs, refactor complex software structures, and provide semantic understanding has transformed both academic research and industrial software development. Before exploring concrete examples of LLM-based applications, this study introduces the fundamental role of LLMs in programming workflows, setting the stage for detailed examination of real-world use cases and systems. The following sections provide categorized examples of how LLMs are applied across multiple areas of programming, including code generation, documentation, debugging, static analysis, testing, and autonomous agent-driven software development.

5.1. Code Generation and Autocompletion

LLM-powered code generation systems synthesize executable code from natural language instructions or partial code fragments. Examples include GitHub [35] Copilot [36], Amazon CodeWhisperer [37] , and Tabnine[38] .

5.2. Automated Code Documentation

Tools like Mintlify Doc Writer [39] and Sourcery AI [40] generate human-readable documentation by analyzing program structure, improving code maintainability.

5.3. Debugging and Error Explanation

Systems such as Copilot Chat[41], Replit AI Debugger[42], and JetBrains AI Assistant[43] interpret error logs and produce actionable debugging steps.

5.4. Static Code Analysis

Semgrep[44] + AI, Snyk Code AI[145], and SonarLint AI enhance traditional rule-based static analysis with semantic reasoning and natural-language vulnerability explanations.

5.5. Test Generation

Diffblue Cover[46], Copilot Test Generation, and TestSigma AI automatically construct unit and integration tests, improving coverage and reliability.

5.6. Semantic Code Search

Sourcegraph Cody[47], Google AI Code Search, and IBM Watson Code Navigator[48] support large-scale repository search using embedding-based semantics.

5.7. Autonomous Bug-Fixing Agents

OpenAI SWE-Agent[49], Repairnator[50], and Meta CodeCompose autonomously detect defects and generate patches.

5.8. DSL Interpretation

Systems like Stripe’s natural-language payment pipelines, NVIDIA Isaac GPT[51], and OpenAI’s function-calling architecture translate domain-specific commands into executable code.

5.9. Refactoring Assistants

Sourcery Pro[40], JetBrains AI Refactor, and Copilot Refactor Commands rewrite codebases for better structure, performance, and maintainability.

5.10. Multi-Agent Systems

OpenAI Reflexion Agents, DevRev AI Engineer Stack[52], and AutoGPT multi-agent frameworks divide software engineering tasks into specialized agent roles.

5.11. Robotics and Embedded Systems

Skydio AI planners[53], NVIDIA Isaac Robotics GPT[51], and Boston Dynamics AI integrations[54] use LLMs for control code and mission planning.

5.12. LLM Research Conclusion

LLMs are redefining modern software engineering, enabling applications ranging from semantic code search to autonomous program repair. Their adoption is rapidly shaping the future of programming practice and research.

6.Effectiveness and Impact of Large Language Models in Computer Programming

The integration of Large Language Models (LLMs) into computer programming workflows, programming education, and automated software development pipelines has produced substantial and measurable impacts. A growing body of empirical research demonstrates that LLMs fundamentally reshape how developers learn, generate, debug, and maintain code. Their influence spans three primary domains:

1. **student learning outcomes**
2. **instructional efficiency and curriculum enhancement**
3. **software development performance, accuracy, and reasoning capability.**

Each domain provides strong evidence of the transformative potential of LLMs in both educational and industrial contexts.

6.1. Impact of LLMs on Student Learning Outcomes

One of the clearest indicators of LLM effectiveness in programming is the significant improvement in student academic performance. According to findings from

Exploration of Computer Programming Teaching Reform Based on Large Language Models , students in LLM-supported classrooms achieve substantially better outcomes than those receiving traditional instruction. The study reports that the experimental class using an LLM-integrated teaching system achieved an average score of 85, compared to 76 in the control group. [35] Additionally, the distribution of student performance shifted positively: there were more high-achieving students and significantly fewer low-performing students [36] in the experimental cohort.

These results are notable for two reasons. First, they demonstrate that LLMs help students understand programming concepts more effectively by providing real-time clarification, step-by-step explanations, and immediate error correction. Second, they show that LLMs can serve as adaptive tutors, capable of personalizing instruction based on individual student needs.

The same study highlights that real-time error detection reduces cognitive overload and prevents students from developing misconceptions. The authors write that LLM-based systems "help students find and correct programming errors in time [37]," which improves conceptual understanding and prevents the accumulation of uncorrected mistakes. This immediate feedback loop mirrors one-on-one tutoring, long considered the most effective instructional method in computer science education.

Furthermore, the personalized learning suggestions generated by LLM-driven learning systems enhance student engagement and motivation. The research reports that students received adaptive exercises and recommendations based on their performance, "further improving the learning effect [38]." Personalized learning is a key marker of modern AI-supported pedagogy, and LLMs appear uniquely capable of delivering tailored programming guidance at scale.

6.2. Impact on Teaching Efficiency and Educational Workflows

LLMs not only benefit students but also significantly reduce instructor workload. One of the most compelling quantitative findings reported in the literature concerns teaching efficiency. In the same study above, researchers document that grading time for programming reports decreased from 150 minutes to only 5 minutes [39], representing a

30× improvement in efficiency. This dramatic time reduction was achieved through automated evaluation, code correctness checks, and feedback generation powered by LLMs.

This efficiency gain has multiple implications:

6.2.1. Reallocation of Instructional Resources

With routine grading tasks automated, instructors can dedicate more time to:

- course design,
- mentorship,
- curriculum development,
- helping struggling students, and
- preparing more advanced or interactive classroom activities.

This shift from administrative tasks to higher-value pedagogical work directly improves course quality.

6.2.2. Consistency and Objectivity of Assessment

LLMs produce:

- consistent scoring,
- standardized feedback, and
- unbiased evaluation.

Human grading is subject to fatigue and inconsistency, whereas automated LLM grading follows the same criteria uniformly.

6.2.3. Enhanced Classroom Participation

The teaching reform study reports that the availability of immediate LLM-based assistance “improves students’ participation [40]”, as they feel more confident seeking help, iterating on their code, and engaging with challenging concepts.

This increase in participation is crucial in programming courses, where students often hesitate to ask instructors for repeated clarifications.

6.3. Impact on Code Generation Accuracy and Programming Performance

A second major body of research evaluates how effectively LLMs generate code for real programming tasks. The findings from Enhancing Computer Programming Education with LLMs provide extensive quantitative evidence. GPT-4 and GPT-4o consistently outperform open-source models such as LLaMA-3 8B and Mixtral-8x7B across multiple metrics, including correctness, execution time, and code quality.

6.3.1. High Pass Rates on Programming Tasks

As shown in Table 2 of the study:

- GPT-4 achieves 99% accuracy [42] across most prompting strategies.
- GPT-4o achieves 100% accuracy [42] using the multi-step prompting strategy.

These near-perfect pass rates indicate that state-of-the-art LLMs can reliably generate functionally correct code for a wide range of LeetCode-style tasks.

This demonstrates that LLMs are not merely autocomplete engines they possess robust problem-solving capability grounded in algorithmic reasoning.

6.3.2. Code Quality and Maintainability

Pylint scores demonstrate that the code produced by LLMs is not only correct but also structurally sound. GPT-4's highest Pylint score reached 9.66 [43], indicating strong adherence to Python programming conventions such as naming standards, formatting practices, and modularization. High Pylint scores correlate with:

- code readability,
- maintainability,
- correctness,
- and long-term project sustainability.

This implies that LLMs can generate production-level code, not just rough prototypes.

6.3.3. Prompt Engineering Effects on Performance

LLM performance improves significantly when aided by structured prompting:

- Base prompts yield 30% success [45] on USACO tasks.
- Multi-step prompts yield 55% success [45].
- Multi-step + specific instructions yield 75% success [45].

These findings show that LLMs can solve far more complex problems beyond typical training data when given:

- iterative reasoning steps,
- pseudo-code generation,
- logical verification stages, and
- domain-specific context.

This supports the view that LLMs already possess latent reasoning abilities that can be "unlocked" with carefully engineered input formats.

7. Impact on Complex Problem Solving and Algorithmic Reasoning

Although earlier studies focused on basic programming tasks, recent research demonstrates that LLMs are increasingly effective in advanced, competition-level challenges. The multi-step prompting framework enables the model to break down complex tasks into:

- pseudo-code,
- validation steps,
- test-case analysis,
- edge-case reasoning,
- iterative refinement, and
- final code production.

This approach mimics how expert programmers think, showing that LLMs can serve as scaffolding systems for developing problem-solving skills. The authors emphasize that such strategies “empower LLMs to guide students through complex problem-solving processes [12],” making them useful even for advanced learners preparing for algorithm competitions or high-level computing tasks.

8. Broader Educational and Computational Impact

Across all studies, several broad impacts emerge:

8.1. Increased Accessibility

Students who previously struggled with programming can now receive instantaneous explanations, interactive examples, and personalized debugging support. This democratizes programming education.

8.2. Improved Engagement and Motivation

Immediate feedback increases student confidence, reduces frustration, and encourages experimentation critical for learning programming effectively.

8.3. Scalability

LLMs allow instructors to support much larger classes without sacrificing personalization.

8.4. Enhanced Professional-Grade Development

Generated code meets professional standards of style and structure, meaning LLMs are increasingly viable in industry settings for rapid prototyping and code augmentation.

9.Conclusion

Research overwhelmingly supports the conclusion that LLMs produce significant positive impacts in computer programming education and software development. They improve student learning outcomes, dramatically increase teaching efficiency, provide high-quality code generation, and enhance both foundational and advanced problem-solving abilities. Their scalability, adaptability, and precision indicate a transformative shift in the future of programming practice one in which LLMs serve as both educators and collaborators.

10. Frontend Executive Summary

Frontend designs for AI-assisted programming platforms predominantly center on chat-based interfaces integrated with a dedicated code environment. This design pattern is pervasive in systems ranging from commercial chatbots to specialized tutors. While many cutting-edge approaches leverage Large Language Models (LLMs) to generate UI layouts or content components, the underlying architectural definition often remains basic. Some contemporary research supports using React-based chat-bot front-ends as a natural choice for AI tutors [67], while other work uses HTML/JavaScript-based generative interfaces for tutor authoring [66]

From a pedagogical perspective, empirical studies show that a well-designed UI significantly influences student outcomes, demonstrating a positive effect on error resolution, anxiety reduction, motivation, and interaction. For example, the use of AI hints has been linked to lower levels of "confrustion" (confusion and frustration) and higher levels of focus [68].

Despite the proven benefits and ongoing technological refinement, a gap exists in the systematic documentation and evaluation of comprehensive, modern frontend stacks. Few studies detail the combined deployment of advanced tools necessary for a scalable, industrial-grade application. This project addresses this critical gap by implementing a robust stack featuring TypeScript, React, Next.js, Monaco Editor, and

MUI. Furthermore, the design explicitly tackles underrepresented non-functional requirements such as security via DOMPurify, seamless streaming via Server-Sent Events (SSE), and enhanced developer experience using ESLint/Prettier.

11. UI Architecture Trends in AI-Assisted Programming Tutors

11.1. Web Frameworks and Frontend Stacks

Recent advancements demonstrate the use of generative AI even in UI design. Calò and MacLellan [66] introduced an AI-Enhanced Tutor Builder that uses an HTML/JavaScript frontend and a Domain Specific Language (DSL) to generate tutor interface drafts. This method, which focuses on generating the component layout, suggests that sophistication in interface creation exists, but the stack description remains at the level of HTML/JS, lacking details on modern framework implementation.

In contrast, Roldán-Álvarez and Mesa [67] explicitly detail the use of a React-based chat-bot front-end combined with Python FastAPI, confirming that a React-based chat front-end is a natural choice in state-of-the-art AI tutor systems. Many systems rely on minimal interfaces, such as the standard ChatGPT UI or IDE-integrated assistants. The proposed project follows this trend by adopting React 18 as the UI library, but elevates the implementation through a comprehensive modern ecosystem. This proposed stack includes TypeScript, Next.js (App Router) for optimized routing and streaming capabilities, MUI (Material UI) for accessible components, Monaco Editor for an in-browser VS Code experience, TanStack Query for efficient data handling, SSE for streaming AI output, and DOMPurify for safe rendering.

11.2. Interaction Modalities: Chat-Centric vs Code-Centric UIs

AI assistance tools primarily use two patterns: chat-centric interfaces (such as ChatGPT/Custom GPT tutors) or split-pane environments. The latter, employed by Roldán-Álvarez and Mesa [67], features a conversation chat on the left and corresponding code examples on the right. Generative interfaces may also be used to create component layouts (layout, form components) for the teacher's interface[66].

The current project adopts an integrated approach: the Chat panel (AI tutor) and the Monaco Editor-based code editor are integrated into a single page. The layout utilizes MUI to support features like side-by-side splitting and toolbars. Furthermore, the system uses SSE (Server-Sent Events) to provide a continuous, streaming output for AI responses, creating a fluid, ChatGPT-like experience.

12. UX Patterns for Scaffolding and Feedback

12.1. AI as Mentor vs AI as Solver – UI Reflection

The project's philosophy dictates that the AI must function as a mentor (scaffolding and guidance) rather than a direct solver. Literature emphasizes this approach. Pankiewicz and Baker [68] showed that GPT hints addressing compiler errors were most effective when providing hints and explanations rather than complete solutions, encouraging students to fix their own code.

This pedagogical constraint informs several UX design choices:

Active User Role: AI-assisted pair programming research suggests the AI acts as a partner, requiring the user to remain the active producer [69]. The UI must avoid "magic buttons" that instantly fill in the code.

Mitigating Dependency: Concerns about students developing reliance on AI tools and seeking "complete solutions" necessitate design flows where students are prompted to make their initial attempts first, and AI suggestions are shown later[70].

Pedagogical Presence: The design incorporates elements of "pedagogical presence," focusing on the tutor's tone, questioning techniques, and providing stepwise guidance, all delivered through the UI/UX [71].

12.2. Design Patterns: Hints, Stepwise Assistance, and Error Highlighting

Specific UI patterns are crucial for realizing the scaffolding approach. In the context of errors, a combination of inline error indicators in the Monaco Editor and conceptual explanations provided in the chat can be used. Hints, such as those examined by Pankiewicz and Baker [68], should contain three parts: an explanation of the error, a solution strategy, and an educational element, which is presented as a pop-up after code submission.

For nuanced assistance, the system can offer graduated help, where the initial response is a general hint, and a “request more explanation” button reveals greater detail. The platform can also integrate modes, such as “Do not write my code directly, only provide explanations”, ensuring that the user maintains an active role in code production.

13. Student Perception, Usability, and Trust in AI Frontends

User satisfaction and trust are major factors in the acceptance of AI tools.

Affective and Motivational Impact: Fan et al. [69] found that AI-assisted pair programming significantly increased intrinsic motivation and reduced programming anxiety compared to individual programming. This demonstrates that the UI’s supportive nature, avoiding pressure and non-judgmental feedback, is vital for student engagement.

Concerns and Trust Issues: Students report significant concerns regarding AI accuracy and dependency. Bhatt et al. [70] found that 35.23% of respondents “occasionally” received incorrect or misleading answers from ChatGPT. A notable percentage also expressed worry about dependency (68.4%), misleading guidance (72.6%), and privacy (66.4%).

These concerns mandate specific frontend design principles:

Fostering Trust: The UI should use clear “AI” or “beta” labels and incorporate micro-copy disclaimers near AI outputs, acknowledging potential inaccuracy and encouraging verification.

Mitigating Dependency: UI settings should limit access to complete solutions and enforce workflows where the student must initiate code writing before requesting assistance.

Security and Privacy: The input prompt area should include data warnings and messages cautioning against entering personal information.

14. Gap Analysis: Modern Frontend Stack and Secure, Integrated Editors

14.1. Lack of Explicit Modern Frontend Stacks

Current literature often fails to document the modern frontend technologies required for scalable AI platforms. Many existing studies either omit the stack details or rely on basic implementations such as HTML/JS (as seen in generative tutor interfaces) or minimal React chat front-ends [66] [67].

The proposed project distinguishes itself by detailing a comprehensive, modern stack that is rarely, if ever, systematically discussed together in academic literature. This includes using TypeScript, Next.js for robust routing and SSR, the Monaco Editor for a feature-rich web IDE experience, MUI, TanStack Query, SSE, DOMPurify, and development tools like ESLint and Prettier. This choice ensures an integrated, extensible, and high-performance coding environment.

14.2. Limited Discussion of Security and Content Sanitization

Frontend security, particularly concerning AI-generated content, is a significant, yet largely overlooked, gap in existing literature. Security risks, such as XSS (Cross-Site Scripting) from AI-generated code or HTML, are seldom addressed.

The project explicitly tackles this vulnerability by integrating DOMPurify to sanitize all AI-generated HTML output before it is rendered in the browser. This proactive security layer ensures the platform is technically safer and fills a critical omission in the design of academic AI-tutor interfaces.

15. Conclusion

This literature review establishes that optimal AI tutor design requires an architecture that integrates chat and code environments within a modern web framework. Pedagogically, the UI must support the AI-as-Mentor model, relying on scaffolding and hints to reduce anxiety and promote self-efficacy. Crucially, the system must address user concerns regarding dependency and data accuracy through transparent and well-designed UX patterns.

The proposed platform aligns with and extends these findings by implementing a modern and scalable technical stack: TypeScript, Next.js, React, integrated Monaco Editor, SSE, and DOMPurify. This architecture not only promises a fluid and optimized user experience but also delivers necessary security and maintainability features, positioning the project to fill documented technical and security gaps in the current literature.

16. Executive Summary

This review examines state-of-the-art (SOTA) systems in AI-assisted programming education published between 2023 and 2025. The analysis focuses on technical architectures (backend/database), AI methodologies, and quantitative performance metrics. The literature reveals a strong industry consensus on **Python-based backends** and **"scaffolding" pedagogies** (providing hints rather than solutions). Crucially, the review identifies a significant gap in current research regarding cost-effective scalability, which the proposed project addresses through the novel integration of **Local LLMs (Ollama)**.

17. Technical Architecture Trends

17.1. Backend Frameworks

The literature demonstrates a clear standard for backend development in Intelligent Tutoring Systems (ITS), driven by the need for seamless integration with AI libraries.

Python Dominance: The majority of successful implementations utilize Python micro-frameworks due to their native compatibility with AI SDKs.

- **Flask:** Widely used for bridging frontend interfaces with prompt engineering logic, as seen in the *Tutor Builder* system by Calò & MacLellan. [71]
- **FastAPI:** Preferred for systems requiring high-performance asynchronous processing, such as the *Intelligent Deep-Learning Tutor* developed by Roldán-Álvarez & Mesa. [72]

Strategic Alignment: The proposed project's selection of **Django/Flask** is strongly supported by these findings, offering superior compatibility compared to Node.js alternatives for AI orchestration.

17.2. Data Management and Persistence

Educational platforms require flexible data structures to handle unstructured data such as chat logs, code snapshots, and affective state tracking.

NoSQL Adoption: Non-relational databases are the standard for storing conversational history. Roldán-Álvarez & Mesa successfully utilized **MongoDB** to store student queries, system answers, and user satisfaction scores [72] .

Strategic Alignment: The proposal's consideration of **MongoDB** aligns perfectly with SOTA approaches for handling the diverse, unstructured data generated during student-AI mentorship sessions.

18. Performance and Pedagogical Validity

18.1. "AI as Mentor" vs. "AI as Solver"

A critical aspect of the proposed project is restricting AI to a "guidance" role. The literature provides empirical evidence validating this approach over direct solution generation.

- **Scaffolding Success:** Phung et al. demonstrated that systems designed to generate **hints** (scaffolding) rather than direct fixes achieve outcomes comparable to human tutors. Their *GPT4HINTS* system achieved **~95% precision** in feedback quality [73] .
- **Reduced Anxiety:** Fan et al. found that AI-assisted pair programming significantly reduced programming anxiety ($p < .001$) and improved intrinsic motivation compared to individual work [74] .
- **Detection of Misuse:** Karnalim et al. highlighted the need for detecting AI misuse, developing a system that identifies "code anomalies" with **89% precision** in controlled environments, reinforcing the need for platforms that control AI output [75] .

18.2. Quantitative Impact on Student Success

Reviewed case studies confirm that properly integrated AI tools lead to measurable academic improvement.

- **Grade Improvement:** Timcenko reported that in a class using AI assistance, the average grade rose to **7.67** (on a 7-point scale), compared to **6.08** in previous years without AI [74].
- **Performance Scores:** Fan et al. reported that students using AI assistance outperformed individual programmers with mean performance scores of **~84 vs. ~75** ($p < .001$) [71].
- **Efficiency:** For instructor-facing tools, Calò & MacLellan showed that AI assistance reduced the time required to build complex tutor interfaces by **68%** [75].

19. Gap Analysis: The Case for Local LLMs

While the literature demonstrates the effectiveness of AI tutors, it also highlights a significant barrier to widespread adoption: **Reliance on Proprietary APIs.**

- **The Cost/Privacy Barrier:** Most reviewed studies, including Phung et al. [72] and Pankiewicz & Baker [76], rely on the **OpenAI API (GPT-4)**. This introduces high operational costs and potential data privacy concerns, limiting scalability for educational institutions.
- **The Proposed Innovation:** The project proposal addresses this specific gap by introducing **Local LLM integration (Ollama)**. By enabling the use of open-source models like Llama or Mistral, the proposed platform offers a **sustainable, cost-effective, and privacy-compliant** alternative to the API-dependent systems found in current research.

20. Database/Backend Summary

The proposed **"AI-Assisted Programming Platform"** is an advancement of the current body of research. It aligns with state-of-the-art findings by adopting the **Python/NoSQL** architecture [73][75] and the **"scaffolding" pedagogy** [72]. Furthermore, it innovates by solving the critical **"accessibility vs. cost"** dilemma identified in the literature through the support of **Local LLMs**.

21.Referances

[1] "Untangling deep learning from artificial intelligence and machine learning."

[2] A. Vaswani *et al.*, "Attention Is All You Need."

[3] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." [Online]. Available: https://github.com/tensorflow/tensor2tensor

[4] H. Touvron *et al.*, "LLaMA: Open and Efficient Foundation Language Models," Feb. 2023, [Online]. Available: http://arxiv.org/abs/2302.13971

[5] J. Kaplan *et al.*, "Scaling Laws for Neural Language Models," Jan. 2020, [Online]. Available: http://arxiv.org/abs/2001.08361

[6] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback."

[7] T. B. Brown *et al.*, "Language Models are Few-Shot Learners." [Online]. Available: https://commoncrawl.org/the-data/

[8] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models Chain-of-Thought Prompting.”

[9] Z. Ji *et al.*, “Survey of Hallucination in Natural Language Generation,” Jul. 2024, doi: 10.1145/3571730.

[10] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the dangers of stochastic parrots: Can language models be too big?,” in *FAccT 2021 - Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, Association for Computing Machinery, Inc, Mar. 2021, pp. 610–623. doi: 10.1145/3442188.3445922.

[11] OpenAI, “GPT-4 Technical Report.”

[12] “Gemini 3 Pro-Model Card Model Information.”

[13] “The Claude 3 Model Family: Opus, Sonnet, Haiku Anthropic.” [Online]. Available: https://docs.anthropic.com/

[14] S. Bubeck *et al.*, “Sparks of Artificial General Intelligence: Early experiments with GPT-4,” Apr. 2023, [Online]. Available: http://arxiv.org/abs/2303.12712

[15] DeepSeek-AI *et al.*, “DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model,” Jun. 2024, [Online]. Available: http://arxiv.org/abs/2405.04434

[16] H. Touvron *et al.*, “LLaMA: Open and Efficient Foundation Language Models,” Feb. 2023, [Online]. Available: http://arxiv.org/abs/2302.13971

[17] M. Liu *et al.*, “Textbook-Level Medical Knowledge in Large Language Models: A Comparative Evaluation Using the Japanese National Medical Examination,” Sep. 12, 2025. doi: 10.1101/2025.09.10.25335398.

[18] M. Yasunaga *et al.*, “LARGE LANGUAGE MODELS AS ANALOGICAL REASONERS.”

[19] Á. Cabezas-Clavijo and P. Sidorenko-Bautista, “Assessing the performance of 8 AI chatbots in bibliographic reference retrieval: Grok and DeepSeek outperform ChatGPT, but none are fully accurate.”

[20] W. H. Walters and E. I. Wilder, “Fabrication and errors in the bibliographic citations generated by ChatGPT,” *Sci Rep*, vol. 13, no. 1, Dec. 2023, doi: 10.1038/s41598-023-41032-5.

[21] Z. Ji *et al.*, “Survey of Hallucination in Natural Language Generation,” Jul. 2024, doi: 10.1145/3571730.

[22] L. Ouyang *et al.*, “Training language models to follow instructions with human feedback.”

[23] G. Fan, D. Liu, R. Zhang, and L. Pan, "The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance," International Journal of STEM Education, vol. 12, no. 16, pp. 1–17, 2025. https://doi.org/10.1186/s40594-025-00537-3

[24] S. Wang, T. Xu, H. Li, C. Zhang, J. Liang, J. Tang, P. S. Yu, and Q. Wen,“Large Language Models for Education: A Survey and Outlook,”*arXiv preprint* arXiv:2403.18105, 2024. https://arxiv.org/abs/2403.18105

[25] Z. Huang, Y. Li, and M. Chen, “Transformer-LSTM Models for Automatic Scoring and Feedback in English Writing Assessment,” 2024. [Online]. Available: https://www.researchgate.net/publication/390938051_Transformer-LSTM_Models_for_Automatic_Scoring_and_Feedback_in_English_Writing_Assessment

[26] R. Roldán-Álvarez and J. Mesa, “Intelligent Deep Learning Tutoring System to Assist Instructors in Programming Courses,”*Applied Sciences*, vol. 14, no. 4115, pp. 1–24, 2024. https://www.mdpi.com/2076-3417/14/9/4115

[27] “Khanmigo,” Khan Academy, Accessed: 2025-11-30. [Online]. https://www.khanmigo.ai/

[28] “What is Duolingo Max?,” Duolingo, Accessed: 2025-11-30. [Online]. https://tr.duolingo.com/help/what-is-duolingo-max

[29] “LearnLM | Google Cloud,” Google, Accessed: 2025-11-30. [Online]. https://cloud.google.com/solutions/learnlm

[30] “OpenStax Assignable,” OpenStax, Accessed: 2025-11-30. [Online]. https://openstax.org/assignable

[31] “K-12 Large-Scale Assessments — Automated Scoring,” Pearson Assessments, Accessed: 2025-11-30. [Online]. https://www.pearsonassessments.com/large-scale-assessments/k-12-large-scale-assessments/automated-scoring.html

[32] [2] C.-Y. Chung, I.-H. Hsiao, and Y.-L. Lin, "AI-assisted programming question generation: Constructing semantic networks of programming knowledge by local knowledge graph and abstract syntax tree," Journal of Research on Technology in Education, vol. 55, no. 1, pp. 94–110, 2023. https://doi.org/10.1080/15391523.2022.2123872

[33] Y. Sun, K. Matsuo, and T. Nakajima “LLM Agents for Education: Advances and Applications,”*arXiv preprint* arXiv:2503.11733, 2025. https://arxiv.org/abs/2503.11733

[34] D. Roldán-Álvarez and F. J. Mesa, "Intelligent Deep-Learning Tutoring System to Assist Instructors in Programming Courses," IEEE Transactions on Education, vol. 67, no. 1, pp. 153–161, Feb. 2024. https://doi.org/10.1109/TE.2023.3331055

[35] https://github.com/

[36] https://copilot.microsoft.com/

[37] https://docs.aws.amazon.com/codewhisperer/

[38] https://www.tabnine.com/

[39] https://www.mintlify.com/

[40] https://www.sourcery.ai/

[41] https://copilot.cloud.microsoft/

[42]
https://replit.com/ai?gad_source=1&gad_campaignid=23286661337&gclid=CjwKCAiA86
_JBhAIEiwA4i9Ju6sRp-tjRGiOwLfnqxpRTX3c8x-
cerLDEU8VTbeptAGooiEbDoAeIxoC70sQAvD_BwE

[43] https://www.jetbrains.com/

[44] https://semgrep.dev/

[45] https://snyk.io/product/snyk-code/

[46] https://www.diffblue.com/diffblue-cover/

[47] https://sourcegraph.com/amp

[48] https://www-ibm-com.translate.goog/products/watsonx-code-
assistant?_x_tr_sl=en&_x_tr_tl=tr&_x_tr_hl=tr&_x_tr_pto=tc

[49] https://openai.com/tr-TR/index/introducing-codex/

[50] https://projects.eclipse.org/projects/technology.repairnator

[51] https://developer.nvidia.com/isaac

[52] https://devrev.ai/careers

[53] https://www.skydio.com/careers

[54] https://bostondynamics.com/

[55] Y. Wang and X. Chen, “Exploration of Computer Programming Teaching Reform Based on Large Language Models,” pp. 4.

[56] Y. Wang and X. Chen, “Exploration of Computer Programming Teaching Reform Based on Large Language Models,” pp. 4.

[57] Y. Wang and X. Chen, “Exploration of Computer Programming Teaching Reform Based on Large Language Models,” pp. 5.

[58] Y. Wang and X. Chen, “Exploration of Computer Programming Teaching Reform Based on Large Language Models,” pp. 5.

[59] Y. Wang and X. Chen, “Exploration of Computer Programming Teaching Reform Based on Large Language Models,” pp. 4.

[60] Y. Wang and X. Chen, “Exploration of Computer Programming Teaching Reform Based on Large Language Models,” pp. 5.

[61] Y. Wang and X. Chen, “Exploration of Computer Programming Teaching Reform Based on Large Language Models,” pp. 5.

[62] Y. Zhao, “Enhancing Computer Programming Education with LLMs: A Study on Effective Prompt Engineering,” Table 2, p. 9.

[63] Y. Zhao, “Enhancing Computer Programming Education with LLMs: A Study on Effective Prompt Engineering,” Table 4, p. 10.

[64] Y. Zhao, “Enhancing Computer Programming Education with LLMs: A Study on Effective Prompt Engineering,” Table 5, p. 10.

[65] Y. Zhao, “Enhancing Computer Programming Education with LLMs: A Study on Effective Prompt Engineering,” Abstract, p. 1.

[66] T. Calò and C. J. MacLellan, “Towards educator-driven tutor authoring: Generative AI approaches for creating intelligent tutor interfaces,” in *Proceedings of the Eleventh ACM Conference on Learning @ Scale (L@S ’24)*, ACM, 2024. https://doi.org/10.1145/3657604.3664694

[67] D. Roldán-Álvarez and F. J. Mesa, “Intelligent deep-learning tutoring system to assist instructors in programming courses,” *IEEE Transactions on Education*, vol. 67, no. 1, pp.

153–161, 2024.
https://doi.org/10.1109/TE.2023.3331055

[68] M. Pankiewicz and R. S. Baker, “Navigating compiler errors with AI assistance: A study of GPT hints in an introductory programming course,” in *Proceedings of the 29th Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE ’24)*, ACM, 2024.
https://doi.org/10.1145/3649217.3653608

[69] G. Fan, D. Liu, R. Zhang, and L. Pan, “The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: A comparative study with traditional pair programming and individual approaches,” *International Journal of STEM Education*, vol. 12, 16, 2025.
https://doi.org/10.1186/s40594-025-00537-3

[70] V. Bhatt, Z. Yu, Y. Hou, and J. Jin, “ChatGPT as a programming tutor: Student perceptions, effectiveness, and challenges,” in *2025 IEEE Global Engineering Education Conference (EDUCON)*, IEEE, 2025.
https://doi.org/10.1109/EDUCON62633.2025.11016463

[71] T. Calo and C. MacLellan, “Towards Educator-Driven Tutor Authoring: Generative AI Approaches for Creating Intelligent Tutor Interfaces,” in *L@S 2024 - Proceedings of the 11th ACM Conference on Learning @ Scale*, Association for Computing Machinery, Inc, Jul. 2024, pp. 305–309. doi: 10.1145/3657604.3664694.

[72] D. Roldan-Alvarez and F. J. Mesa, “Intelligent Deep-Learning Tutoring System to Assist Instructors in Programming Courses,” *IEEE Transactions on Education*, vol. 67, no. 1, pp. 153–161, Feb. 2024, doi: 10.1109/TE.2023.3331055.

[73] T. Phung *et al.*, “Automating Human Tutor-Style Programming Feedback: Leveraging GPT-4 Tutor Model for Hint Generation and GPT-3.5 Student Model for Hint Validation,” in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Mar. 2024, pp. 12–23. doi: 10.1145/3636555.3636846.

[74] G. Fan, D. Liu, R. Zhang, and L. Pan, “The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: a comparative study with traditional pair programming and individual approaches,” *Int J STEM Educ*, vol. 12, no. 1, Dec. 2025, doi: 10.1186/s40594-025-00537-3.

[75] O. Karnalim, H. Toba, and M. C. Johan, “Detecting AI assisted submissions in introductory programming via code anomaly,” *Educ Inf Technol (Dordr)*, vol. 29, no. 13, pp. 16841–16866, Sep. 2024, doi: 10.1007/s10639-024-12520-6.