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 offers an analytical overview of the development and application of Large Language Models (LLMs) across 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 illustrate how these models differ in terms of efficiency, reasoning capability, and applicability across real-world scenarios.

In the context of education, the review examines how LLM-based tools are being integrated into teaching and learning practices, highlighting specific examples and evidence of their impact 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 the related technical components, including frontend, backend, 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.

1. Fundamentals of Generative AI and LLMs

1.1. The Hierarchy of AI

Artificial Intelligence (AI) evolution could be described as a layered progression of technology. In general, AI is designed to replicate analytic functions that are similar to humans; the most common examples are learning and decision making. Machine Learning (ML) is a must to train AI models from given data. And subset of ML, Deep Learning, relies on layered artificial neural networks that emulate the structure of the human brain [1]. Artificial neural networks open the gates of the generative systems. GenAI marks the shift from traditional “discriminative” models, which focuses on analyzing or classifying existing data, to models that can autonomously create new text, images, or code that is realistic, coherent, and novel.

1.2. Transformer Architecture

Transformer Architecture was one of the most important things to accelerate this transformation [2]. Self-attention mechanism enables models to simultaneously evaluate relationships between all words in a sentence, instead of processing them separately was the key innovation behind Transformers. The industry changed in 2017 when Transformer architecture was introduced. Instead of processing words one after another, the model uses a 'self-attention' mechanism to analyze the whole sentence at the same time to find connections. Similar to how the reader stresses certain words on specific words to understand intent, this design helps the model connect distant ideas and grasp context. Systems like BERT and the GPT family [3] used this foundation to grow massively, handling complex language tasks with high accuracy.

1.3. Pre-training, Scaling Las, and Efficiency

Large Language Models (LLM) are using a large quantity of data to predict the next token in the sequence. Before training we feed these models with trillions of tokens from diverse seourcves like CommonCrawl, Wikipedia, GitHub, and scientific papers. For example, LLaMa series demonstrated high quality data and effective training strategies that can allow smaller models to outperform much larger ones [4], Breaking the assumption of larger models gives better solutions. We know from scaling laws that the more compute and data we have, the better our AI models perform [5]. Still, getting these models out the door means we need to focus more and more on how fast and efficient they are when they're actually being used.

1.4. Alignment and Reinforcement

While a pre-trained LLM is a more sophisticated form of a text generator, it is not naturally aligned with the same human values or intent. In order to create safer, more cooperative and reliable models, RLHF is used. RLHF is an iterative process that starts with SFT, teaching a model through human written examples to demonstrate desired behavior [6]. Next, a Reward Model is created using human ranked responses to determine which responses were preferred. Finally, RLHF adjusts the model's behavior through the use of reinforcement learning algorithms, such as PPO, to maximize the reward signals received from the Reward Model to provide the user with better, more truthful and less harmful responses.

1.5. Prompt Engineering Techniques

Prompt Engineering - How Users Interact with LLMs- also plays a major role in determining the performance of a model. Zero-Shot Prompting and Few-Shot Prompting are two of the most common techniques used to direct the model to respond correctly by either solely relying upon prior knowledge [7] or providing an example of the correct response. Chain of Thought Prompting allows the model to divide complex reasoning tasks into intermediate steps [8], dramatically enhancing results when used intentionally. Role-Prompting directs the model to respond to a given question or task as if it were responding as a specific professional identity or persona, helping to define the tone, format, and domain relevance of generated responses.

1.6. Limitations

As powerful as LLMs have become, they still contain numerous inherent limitations. Hallucinations can occur in an LLM and produce confident yet factual inaccuracies [9] since LLMs generate text based on statistical patterns and do not rely on established facts. Biases that exist within the training data can be replicated or amplified through an LLM, potentially resulting in inaccurate or biased responses [10]. As a result of the lack of transparency, understanding exactly how a model arrived at its response is often impossible and highlights the need for continued research in areas of transparency, alignment and safe and responsible deployment of AI Systems.

2. Lead LLM Products

Currently, there are a variety of Large Language Models available; each family has been optimized to meet different trade-offs regarding the speed, cost, and reasoning ability of a model.

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)

The Gemini 3 family of models, including the powerful Gemini 3 Pro, is built for complex, real-world tasks. Its main advantage is advanced multimodal understanding across text, images, video, and code. Gemini 3 Pro stands out for its high-accuracy reasoning, capacity to process massive amounts of data (up to 1 million tokens), and ability to handle multi-step problems. Critically, these models are designed for efficiency using a specialized architecture, making them excellent, benchmark-leading tools for integrated AI systems [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 [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 citations 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, supply-chain decision-making, and academic reliability.

3.1. Performance in Medical Domains

Results from the study indicate that all four models have reached a high level of medical proficiency, consistently scoring over 94% across both general and clinical categories. General knowledge questions saw the highest performance, with some models reaching 99.0% accuracy, but clinical reasoning capabilities were also strong. Gemini 2.5 Pro led the clinical category specifically with a 97.0% success rate. While clinical questions are generally more difficult than basic factual recall, the performance gap between these two areas was notably small for most of the models. This suggests that the current generation of LLMs has effectively mastered both foundational knowledge and complex diagnostic decision-making [17].

3.2. Supply Chain, Decision Making

This domain evaluates models on progressively more complex tasks: theoretical multiple-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

Some recent studies point out an interesting pattern: simply telling a model to "think step by step" (implicit CoT) doesn't necessarily help. In fact, for some models, especially Claude, it can actually make things worse on harder problems. But when the model is required to write out its reasoning in full (explicit CoT), the accuracy jumps noticeably. One example that's often mentioned is DeepSeek, which goes from getting none of the medium-level numerical questions right to solving around 80% of them once it has to show its work [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 complete GPT-5 model.

3.3. Academic Reliability & Hallucination Rates

A key takeaway from this research is that AI chatbots are consistently making up references, which is a major, built-in problem. This "hallucination" happens when the model can't find a real source in its training data, so it just invents one that sounds academic and correct.

It's not random behavior; it's how these systems are designed. They prioritize creating smooth, natural-sounding sentences over checking if the facts or in this case, the sources are actually real. The result? They generate fake citations that look totally legitimate, mixing real-sounding authors and standard journal names into something that's really hard to spot as false.

We saw huge differences between models. Some, like Grok and DeepSeek, were perfect, but others, especially Copilot, Perplexity, and Claude, were big offenders, sometimes creating entire lists of made-up references. The most worrying issue is when they create "structured fakes," taking a real author or publisher and attaching them to a fake article title. This makes the bad citations look credible at a glance, posing a huge risk to students who might use them without checking.

The type of source matters, too. Models were much more likely to fake journal articles than textbooks. Textbooks are common in their data, but specialized or new journal articles are not. This is why fields focused on articles, like Engineering and Health Sciences, saw more fakes. Sometimes, the chatbots even tried to impress the user by adding recent publication dates (like 2020–2025) to entirely fabricated sources, proving their main goal is plausibility, not accuracy.

Ultimately, hallucination is a fundamental flaw when LLMs are used for academic tasks requiring precise citations. Even when they give a real source, they often introduce small errors. This all highlights one thing: You must manually check every single reference. We need better training data, better integration with databases, and better education to make sure students use these tools responsibly [19].

3.3.1. Fabrication Rates

Grok and Deepseek show perfect integrity, with all references verified. On the other hand, Copilot shows some issues. ChatGPT and Gemini often generated “hallucinated” citations that appeared plausible by giving authors articles they did not work on [20].

3.3.2. Document Type Differences

Compared to journals models are documented to be far more reliable with book referances [21]. This shows LLMs are struggling with details of academic literature. Compared to books, academic articles are far more numerous and specific and appaering far freaquently, whereas books are treated as distinct, major entities within model’s training data. But these models are often learned with the rebuilded version of the source; they hallucinate citations by pairing real, established authors with non-existing article titles. This explains why specific articles’ error rates are high, making most models unreliable for deep literature reviews without human touch.

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 determinants of performance than raw parameter count alone.

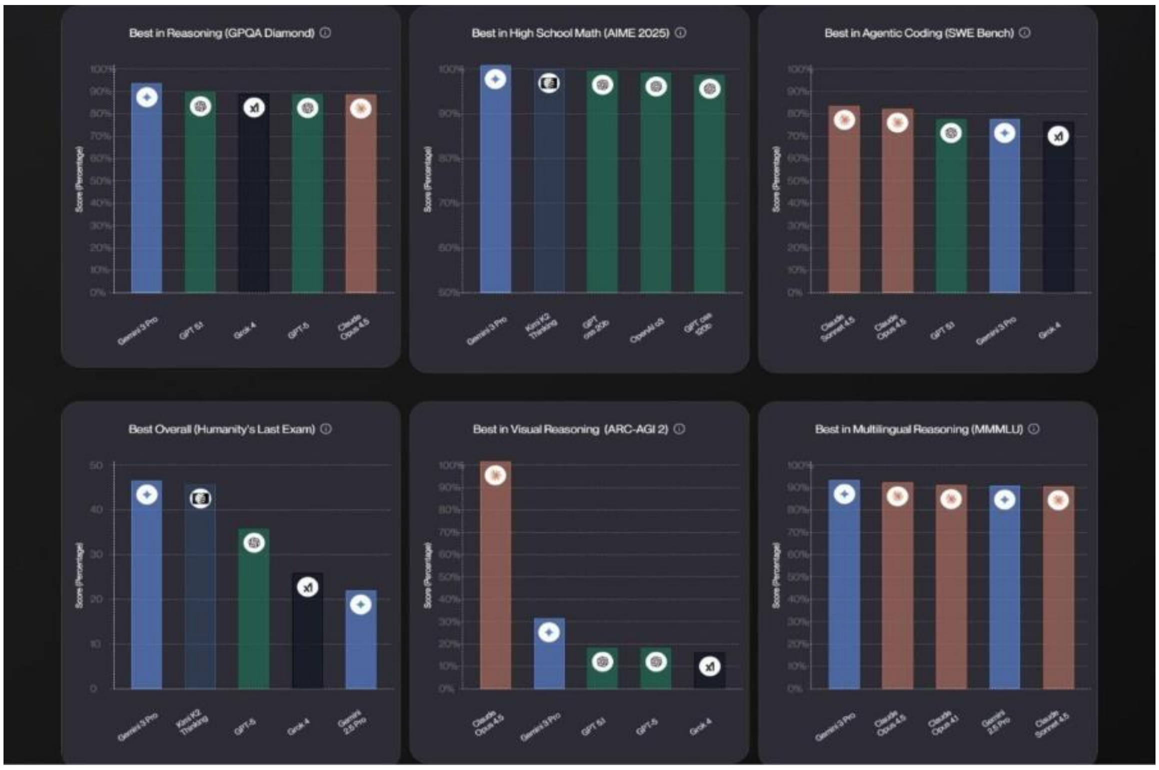

Figure 1: LLM performance benchmarks (Reasoning, Coding, and Overall Quality) compare various large language models. Data collected by Vellum AI, 2025. [https://www.vellum.ai/llm-leaderboard?utm_source=google&utm_medium=organic]

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 increasingly important in the broader shift in educational technology, and their influence is growing as the field continues to evolve. Although they are built on Transformer-based architectures, which give them strong performance in natural language tasks, what makes them stand out is their ability to interpret context in a more layered way and to offer explanations that connect reasonably well with instructional aims. Instead of simply processing text, they can shape their output to support learning goals, a capability that has made them far more than mere technical language tools. Traditional teaching methods generally rely on a one-way transfer of information, whereas systems that use LLMs can adjust themselves almost instantly to how quickly a student is learning, what they might need at a given moment, or where they seem to be misunderstanding something. This capacity to change their responses on the fly allows LLMs to act as flexible learning companions and also makes them scalable tools in instructional settings. Their introduction into educational environments has noticeably altered how students and instructors interact, since the models do more than generate replies; they can explain concepts in ways that reflect basic pedagogical thinking and can point out when a learner is struggling with an idea, which enables them to function as active contributors to the teaching process rather than simple answer-producing systems. Research in this field indicates that the individualised feedback provided by LLMs plays a notable role in enhancing learners' motivation and improving their self-regulation abilities [23], [24].

From the teachers' perspective, LLMs are increasingly 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 students to acquire incorrect information. There is also the frequently debated concern that excessive student engagement with these systems might gradually erode the fundamental capacity for independent problem-solving, a dependency that leaves them with what can only be described as a rather shallow command of the topic. In fact, this worry isn't just something people talk about in theory. A study published in Applied Sciences [26] looked into the issue, and the authors noticed that students who depended on LLMs quite a lot didn't do as well in their courses. It's not a dramatic claim, but it does show that the negative effects of overuse are already showing up in real classrooms, which means the problem isn't as abstract or distant as it might seem at first. For this reason, it is important to integrate LLMs into learning settings in a measured and intentional way, ensuring their use supports rather than replaces essential cognitive and pedagogical practices. The integration of LLM-based solutions into education raises many ethical considerations. Since these systems are trained on large, highly mixed datasets, a number of problems can arise, privacy being one of them, along with the risk of reinforcing biases or even spreading misinformation. Because of this, institutions need to be clear about how they gather, use, and store student-related data, and monitor these processes. As noted in [24], having such rules in place isn't optional anymore; it's really the basic groundwork for using these tools safely.

Findings in the literature reveal that LLM-based systems offer an effective structure that significantly supports teaching processes. These technologies can be quite helpful in education, whether it's by supporting students as they make sense of new concepts, reducing the routine work teachers have to do, or providing learners with feedback that actually fits their needs. Still, using LLMs effectively requires more than just relying on what the technology can do; it also means grounding their use in sound pedagogical thinking and keeping the learning experience centred on human judgment and interaction.

4.1. Example Projects and Applications in General Education

In many educational settings, tools built on LLMs have grown into a wide-ranging set of applications that can adjust to different kinds of learners and scale easily when needed. Rather than simply delivering information, these systems create learning environments that can organise the process, offer guidance, and shift their support depending on what each student requires at a given moment. Along with digitalisation, the primary goal of LLM-based platforms is to reduce teacher workload, increase learning speed, and create sustainable teaching models suitable for learners of different levels. In this context, this section details how institutional, commercial, and academic systems are positioned in educational processes.

4.1.1. Khanmigo (Khan Academy)

A useful illustration of how LLMs are being incorporated into everyday education is Khan Academy's Khanmigo, which is powered by GPT-4. Rather than just displaying the correct answer, the system encourages students to work through the problem themselves by breaking the task into smaller steps. Sometimes a student gets stuck because they begin solving a problem in a way that won't really work. In these cases, Khanmigo does not give a full explanation right away. Instead, it offers a brief hint or asks a quick question to help the student see what isn't working and try a new approach. By working in this supportive yet non-intrusive manner, the system encourages students to rely on their own thinking while preventing them from becoming overwhelmed, ultimately helping them form a deeper, longer-lasting understanding [24][27].

4.1.2. Duolingo Max

Duolingo Max is a leading example of how large language models can be used in language learning. Its AI dialogue system lets learners have real-time conversations in everyday language. If a student makes a mistake, the system corrects it and explains the grammar and meaning to help them get better. Such explanatory feedback strengthens lasting comprehension in language learning, while the system's role-playing scenarios 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 customised 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 analyses data from completed activities and identifies individual learning gaps. The system identifies areas where students struggle and creates personalised learning plans. This aspect delivers a considerable 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)

Automated assessment and feedback systems are also common in education. Organisations like ETS and Pearson have created tools that review student writing using rubrics and assess things like content structure, logical flow, and accuracy. These systems help teachers by speeding up grading and making it more consistent, especially in writing-intensive classes and large exams [25]. They also give students clear feedback on how to improve, not just a score [31].

4.1.6. PQG Systems (AST + Concept Graphs)

Programming Question Generation (PQG) systems help teachers in programming education by using Large Language Models (LLMs) to create coding questions that fit each student's skill level. These systems use tools such as Abstract Syntax Trees (AST) and Local Knowledge Graphs (LKG) [32] to make sure the questions match the curriculum. This means teachers spend less time making new questions and can give students more practice on topics they find challenging.

4.1.7. Multi-agent LLM Systems

Multi-agent LLM systems use several artificial intelligence agents that share tasks. For example, one agent checks a student's solution and finds mistakes, another gives feedback, and a third updates the learning plan as the student improves. Studies show that these systems often give more consistent and reliable educational results than 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 personalised 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 on the impact of LLM-based technologies in education examines learning outcomes, cognitive processes, motivation, social interaction, assessment accuracy, and instructional design. These studies show both the benefits and drawbacks of using LLMs in learning.

4.2.1. Learning Performance Impact

Research shows that students make real progress when they use LLMs for explanations, especially in STEM and programming. LLMs guide them step by step and help them understand complex ideas. In programming, students can spot errors more easily, understand code structure, and learn faster. Results from the Intelligent Deep Learning Tutoring System also show that LLM-style feedback improves students' problem-solving and strongly supports their understanding of concepts [26].

4.2.2. Affective (Motivational & Emotional) Effects

The affective dimension is a prominent area 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 found that LLM-supported paired programming increases both intrinsic motivation and perceptions of benefits in the learning process among students [23]. 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 [23]. 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 across content integrity, structure, consistency, and rubric alignment [25]. In large exams, these systems help teachers grade faster, make it easier to analyse student writing, and cut down on grading differences between people.

4.2.5. Negative Effects & Over-reliance Issues

Research shows that using LLMs does not always lead to positive outcomes. In a study published in Applied Sciences, Roldán-Álvarez and Mesa (2024) found that students who depended too much on LLMs had lower achievement [26]. The Spearman correlation results showed these students had less conceptual depth and tended to learn more superficially. This suggests that LLMs should be used in moderation in learning.

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]. Different roles, like expert, critical, and explanatory agents, help at each stage of learning and make the system stronger.

4.2.7. Limitations, Risks, and Ethical Concerns

Some studies point out that LLMs have limitations. These models may misinterpret context and sometimes generate inaccurate or false information, which can cause misunderstandings or reinforce students' misconceptions. There are also concerns about data privacy, systemic biases, and ethical responsibility, especially when student data is stored and processed [24] [33]. As LLM-based technologies become more common in education, institutions need to provide oversight to keep these systems reliable.

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 well set up, these systems provide strong support for learning and help improve educational quality over time.

5. Applications of Large Language Models in Computer Programming

Big linguistic models, LLMs, these have become super important ,in modern programming yes, they interpret contexts, generate code ,fixing bugs reshaping complex software, so much, they've changed, they really have, Academic settings or big companies, it's all changing. Before we jump straight into examples which are grounded in the whole LLM scene, we start by setting up the basic territory of these models in code-making workflows, just to prepare us for some intense review, of things happening out there in real programming situations. What follows are types of examples like, how LLMs work in various programming scenes like code making, making things clear in documents, finding glitches, static studies, test creating, and oh, software that just does things on its own.

5.1. Code creation and Helping to fill in the gaps

Systems driven by LLM that make actual running code from just words or bits of already existing code. Um let’s see, yeah, examples, GitHub[34] copilot35], that Amazon thing Code Whisperer[36], also Tabnine[37].

5.2. Automatic creation of code explanations

Tools such as, what are they, Mintlify Doc Writer[38] and that Sourcery AI[39] thing, they sort of read through program setup and really help keep the code usable.

5.3. Sorting out bugs and explaining mistakes

You got things like Copilot Chat[40], that Replit AI Debugger[41], and oh, JetBrains AI Assistant[42], they look through error info and suggest steps to make things right.

5.4. Analysis of code without changing it

There's Semgrep[43] added with AI, Snyk Code AI[44] too, and SonarLint with AI which take the usual rules-based analysis and throw in some clever reasoning and explanations that kind of sound like ordinary talk.

5.5. Making tests

There's Diffblue cover[45], that Copilot thing for tests, and TestSigma with AI, they kind of build tests on their own, making sure things cover more and stay reliable.

5.6. Searching code with meaning

You’ve got Sourcegraph Cody[46], Google doing code search with AI, and IBM’s Watson helping navigate code[47] which uses big repository searches with what they call embedding-based semantics.

5.7. Bugs getting fixed on their own

There's OpenAI's SWE-Agent[48], Repairnator[49], yeah, and Meta's Codecompose, they sort of find problems by themselves and come up with fixes.

5.8. Understanding specific domains

Like Stripe’s thing for payments that you just talk to, NVIDIA’s Isaac GPT[50], and something from OpenAI that turns special commands into something the computer can actually do.

5.9. Refactoring Assistants

Sourcery Pro[39], JetBrains AI refactor, and Copilot refactor commands rewrite codebases, aiming for better structure, performance and maintainability, or so they say anyway.

5.10. Multi-Agent Systems

OpenAI Reflexion Agents, these DevRev AI Engineer Stack[51], and AutoGPT multi-agent frameworks, they sort of separate software engineering tasks into specialized agent roles, you know?

5.11. Robotics and Embedded Systems

Skydio AI planners[52], also something like NVIDIA Isaac Robotics GPT[50], and then there’s Boston Dynamics AI integrations[53], which use LLMs for something to do with control code and mission planning.

LLMs, they're kind of redefining modern software engineering the way we see it, applications ranging from what you would call semantic code search to, let’s say, fixing the programs on their own. Their adoption, it’s sort of making waves, really influencing what's next for programming practice and even research I'd say.

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 had substantial and measurable impacts. There is a growing body of empirical research indicates LLMs fundamentally change how developers learn, create, debug, and maintain code, Their influence spans three main domains:

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

Each domain demonstrates 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 how effective LLM are in programming, is the significant increases in student academic performance, really. According to findings from an exploration of Computer Programming Teaching Reform Based on Large, Language Models. Students in classrooms supported by LLMs achieve substantially better outcomes. Than those with traditional instruction. The study pointed out that the experimental class using an LLM-integrated teaching system got an average score of 85 compared to 76 in the control group[54]. But also, the distribution of student performance shifted positive[54]. Like more high-achieving students, and significantly fewer low-performing students in the experimental cohort.

These results, they are notable for two main reasons. First, they show LLMs help students get programming concepts more effectively by giving real-time clarification, explanations step by step, and immediate error corrections. Second, they show that LLMs can behave as adaptive tutors that's very capable of personalizing instruction based on individual student needs, see?

The same study highlights real-time error detection, which lowers cognitive overload and prevents students from developing wrong ideas. The authors write that LLM-based systems "help students find and correct programming errors in time[54]",this really improves understanding of concepts and prevents the accumulation of uncorrected mistakes. This immediate feedback loop is kind of like one-on-one tutoring, which has always been viewed as the most effective instructional method in computer science education, so to speak.

Moreover the Personalized learning suggestions that come out of LLM-driven learning systems improve student engagement and motivation quite a bit. The research reported that students received adaptive exercises and recommendations based on their performance, which "further improve the learning efficiency.[54]" Personalized learning it's really a key marker of modern AI-supported pedagogy. And it seems like LLMs are particularly equipped to deliver tailored programming guidance on a large scale.

6.2. Impact on Teaching Efficiency and Educational Workflows

LLMs not only help students but also significantly lowers the workload of instructors. One of the most impressive quantitative findings reported in the literature is about teaching efficiency. In the same study, it's noted that grading time for programming reports dropped from 150 minutes to just 5 minutes[54], that's like a 30× improvement in efficiency. This huge decrease in time was due to automated evaluation, checks for code correctness, and feedback generation all powered by LLMs.

The efficiency gain has multiple implications:

Reallocation of Instructional Resources
With automatic grading, tutors can really focus more on course designing, mentoring, curriculum developing, helping students who struggle, and developing more advanced or interactive classroom activities.

Consistency and Objectivity of Assessment
LLMs can give consistent scores and provide standardized feedback, offering unbiased evaluation. Unlike humans—whose grading is affected by fatigue or inconsistency—LLMs follow the same criteria every time.

Enhanced Classroom Participation
The study claims that having instant help from LLM systems "enhances how much students get involved[54]." Students feel more confident when asking for help, retrying activities, and discussing difficult concepts—very important in programming courses.

6.3. Impact on Code Generation Accuracy and Programming Performance

Another significant area of study investigates how well large language models adeptly form code for practical programming tasks. Research presented in enhancing Computer Programming Education through LLMs offers plenty of quantitative evidence. Models like GPT-4 and GPT-4o tend to outperform open-source alternatives such as LLaMA-3 8B and Mixtral-8×7B on various metrics. Correctness, execution time, and code quality receive a lot of attention; GPT versions show better results.

6.3.1. High Pass Rates on Programming Tasks

Table 2 in the study shows:

- GPT-4 achieved around 99% accuracy with most prompting methods[54].
- GPT-4o hit 100% accuracy using the multi-step prompting approach[55].

These results suggest LLMs can solve real programming tasks with extremely high reliability, not just autocomplete code.

6.3.2. Code Quality and Maintainability

Pylint scores show that LLMs create code that is correct and well structured. GPT-4 got a top Pylint score of 9.66[55], meaning it adheres to best practices for style, naming, and organization. This means:

• The code is readable
• The code is maintainable
• The code is correct
• The code is production-quality

LLMs can therefore generate high-quality, industry-ready code, not just rough drafts.

6.3.3. Prompt Engineering Effects on Performance

LLMs do better with structured prompting:

• Basic prompts get ~30% success on USACO tasks[55]
• Multi-step prompts → ~55%
• Multi-step + well-defined instructions → ~75%

This shows LLMs benefit greatly from:

• reasoning scaffolding
• pseudo-code steps
• validation cycles
• richer task context

6.4. Impact on Complex Problem Solving and Algorithmic Reasoning

Past studies focused on simple coding, but recent research shows LLMs now approach competition-level problems. Multi-step prompting lets them break tasks into:

• pseudo-code
• validation checks
• test case analysis
• edge-case logic
• iterative fixes
• final code

This mirrors expert problem-solving and helps students learn higher-level reasoning. The authors stress that these methods "let LLMs help students through tricky problem solving[55]," meaning they support advanced learners effectively.

6.5. Broader Educational and Computational Impact

Across studies, several general impacts emerge:

6.5.1. Greater Usability

Students who struggle with programming can now get immediate explanations and personalized debugging, democratizing programming education.

6.5.2. Enhanced Involvement and Inspiration

Quick feedback reduces frustration, increases confidence, and supports experimentation, critical in learning programming.

6.5.3. Scalability

LLMs allow instructors to manage large classes without losing individualized support.

6.5.4. Improved Professional Development

Since generated code follows professional conventions, LLMs are useful for rapid prototyping and code augmentation in real industry workflows.

7. Applications of Large Language Models in Computer Programming 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 basic and advanced problem-solving abilities. Their scalability, adaptability, and accuracy signal a transformational shift in the future of programming practice , with LLMs acting not only as assistants but as collaborators.

8. Frontend Executive Summary

Frontend designs for AI-assisted programming platforms predominantly center on chat-based interfaces integrated with a dedicated code environment. This design choice is common in systems ranging from commercial chatbots to specialized tutors. Even though cutting-edge approaches leverage Large Language Models (LLMs) to generate UI layouts, the underlying architectural definition mostly remains basic. Some research supports using React-based chat-bot front-ends as a natural choice for AI tutors [26], while other work uses HTML/JavaScript-based generative interfaces for tutor authoring [56]

Empirical studies show that a well-designed UI can greatly influence student outcomes, positively impacting error resolution, anxiety reduction, and motivation from a pedagogical view. For instance, the use of AI hints has been linked to lower levels of confusion and frustration and also higher levels of focus [57].

Despite the benefits and ongoing technological refinement, a gap exists in the systematic documentation and evaluation of modern frontend stacks. Some studies detail the usage of advanced tools necessary for an industrial-grade application. Our project addresses this critical gap by implementing a stack featuring TypeScript, React, Next.js, Monaco Editor, and MUI. Furthermore, the design addresses underrepresented requirements, like security via DOMPurify, seamless streaming via Server-Sent Events (SSE), and an improved developer experience via ESLint/Prettier.

9. UI Architecture Trends in AI-Assisted Programming Tutors

9.1. Web Frameworks and Frontend Stacks

Recent advancements show the use of generative AI even in UI design. Calò and MacLellan [56] 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 depth in interface creation is there, but the stack description remains at the HTML/JS level, lacking details on modern framework implementation.

In contrast, Roldán-Álvarez and Mesa [26] detail the use of a React-based chatbot frontend combined with Python FastAPI, confirming that a React-based chat frontend is a suitable 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 project follows this trend by adopting React 18 as the UI library, but enhances the implementation with a broader modern ecosystem. The stack of the project 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.

9.2. 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 [26], features a conversation chat on the left and corresponding code examples on the right. Generative interfaces may also be used to create component layouts for the teacher's interface[56].

Our project uses an integrated method: the chat panel (AI tutor) and the Monaco Editor-based code editor are packaged into a singular page. The layout uses MUI to support features like side-by-side splitting and toolbars.

10. UX Patterns for Scaffolding and Feedback

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

Our project's perspective dictates that the AI must function as a mentor rather than a direct solver. Pankiewicz and Baker [57] 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:

- AI-assisted pair programming research suggests the AI acts as a partner, requiring the user to remain the active producer [23]. The UI must avoid "magic buttons" that instantly fill in the code.
- 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 [58].
- 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 [59].

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

Specific UI patterns are crucial for making the scaffolding approach. In the context of errors, a combination of error indicators in the Monaco Editor and conceptual explanations in the chat can be used. Hints, such as those examined by Pankiewicz and Baker [57], should include 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", making sure that the user maintains an active role in code production.

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

User satisfaction and trust are major factors in the acceptance of AI tools. Fan et al. [23] found that AI-assisted pair programming truly increased motivation and reduced programming anxiety compared to individual programming. This demonstrates that the UI's supportive nature, avoiding pressure and non-judgmental feedback, can be essential for student engagement.

Students report some concerns about AI accuracy and dependency. Bhatt et al. [58] found that 35.23% of respondents "occasionally" received incorrect or misleading answers from ChatGPT. A considerable percentage also expressed worrying about dependency (68.4%), misleading guidance (72.6%), and privacy (66.4%).

These concerns demand specific frontend design principles:

- The UI should use clear "AI" or "beta" labels and incorporate micro-copy disclaimers near AI outputs, acknowledging potential inaccuracy and encouraging verification.
- UI settings should limit access to complete solutions and enforce workflows where the student must initiate code writing before requesting assistance.
- The input prompt area should include data warnings and messages cautioning against entering personal information.

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

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 [56] [26].

Our 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 routing and SSR, the Monaco Editor for a well-featured web IDE experience, MUI, TanStack Query, SSE, DOMPurify, and development tools like ESLint and Prettier. This choice solidifies an integrated and high-performance coding environment.

13. Frontend 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 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.

Our platform aligns with and extends these findings by applying a scalable technical stack: TypeScript, Next.js, React, integrated Monaco Editor, SSE, and DOMPurify. This architecture promises a comfortable user experience but also delivers necessary security and maintainability features, positioning the project to fill documented technical and security gaps.

14. Executive Summary

This review surveys state-of-the-art (SOTA) systems in AI-assisted programming education between 2023 and 2025. The technical architectures-which mostly exist as backend/database, AI methodologies, and scalar performance metrics in the literature-are analyzed. There is a strong industrial consensus on the use of Python-based backends and pedagogies of "scaffolding" when it comes to hint-providing and not providing the complete solution. Crucially, this review identifies an important gap in present research around cost-effective scalability, which is precisely what the proposed project, through the novel integration of Local LLMs-Ollama-addresses.

15. Technical Architecture Trends

15.1. Backend Frameworks

The literature showcases a clear standard within the development of backends in ITS because of the need for seamless integration with AI libraries.

• **Python Dominance:** Most of the successful implementations are done using Python micro-frameworks since most AI SDKs natively support Python.

o **Flask:** Widely used to connect frontend interfaces with the logic of prompt engineering, such as the Tutor Builder system proposed by Calò & MacLellan. [56]

o **FastAPI** is best used for systems that need high-performance asynchronous processing, as in the case of the Intelligent Deep-Learning Tutor by Roldán-Álvarez & Mesa [26].

• **Strategic Alignment:** The findings presented here severely support choosing Django/Flask for the proposed project, with significantly better compatibility than Node.js alternatives for AI orchestration.

15.2. Data Management and Persistence

Such unstructured data as chat logs, code snapshots, or affective state tracking require flexible data structures necessary in educational platforms.

• **NoSQL Adoption:** Non-relational databases have become the norm when it comes to storing the history of conversations. Roldán-Álvarez & Mesa used MongoDB with much success to store student queries, system answers, and user satisfaction scores [26].

• **Strategic Alignment:** The consideration given in the proposal of MongoDB aligns precisely with SOTA approaches intended to handle diverse and unstructured data generated in student-AI mentorship sessions.

16. Performance and Pedagogical Validity

16.1. "AI as Mentor" vs. "AI as Solver"

Restricting AI to a "guidance" role in the proposed project is critical. Empirical evidence that validates this approach over direct solution generation may be found in the literature.

• **Scaffolding Success:** Phung et al. showed systems designed to generate hints or scaffolding rather than direct fixes obtain performances comparable to human tutors. Their GPT4HINTS system achieved ~95% precision in the quality of feedback [60].

• **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 [23].

• **Misuse Detection:** Karnalim et al. addressed the issue of misuse detection in AI by proposing a system that detects "code anomalies" with 89% precision under controlled environments; this further reinforces the need for platforms to closely regulate AI output.[61].

16.2. Quantitative Impact on Student Success

Case studies reviewed confirm that appropriately integrated AI tools provide 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 [6].

• **Performance Scores:** Fan et al. reported that students using AI assistance outperformed individual programmers with mean performance scores of ~84 vs. ~75 ($p < .001$) [23].

• **Efficiency:** When using instructor-facing tools, Calò & MacLellan demonstrated how AI assistance reduced the time required to build complex tutor interfaces by 68% [56].

17. Gap Analysis: The Case for Local LLMs

While the literature indicates the efficacy of AI tutors, it also points out a very significant barrier to their wide diffusion: dependence on proprietary APIs.

• **Cost/Privacy Barrier:** Most reviewed studies, such as by Phung et al. [60] and Pankiewicz & Baker [57], rely on the OpenAI API, namely GPT-4. This introduces high Operational costs and potential data privacy issues, which also limit scalability for educational institutions.

• **Proposed Innovation:** This gap is filled by the project proposal through the development of an integrated Local LLM, which is referred to throughout as Ollama. The proposed platform allows using open-source models like Llama or Mistral to obtain a sustainable, cost-efficient alternative that is compliant with privacy compared to API-dependent systems in current research.

18. Backend Conclusion

The proposed "AI-Assisted Programming Platform" represents an advance over the existing body of work. It adheres to state-of-the-art results through the use of the Python/NoSQL architecture [60][61] and the "scaffolding" pedagogy [26]. However, it further innovates through solving the critical "accessibility vs. cost" dilemma found in the literature with support for Local LLMs.

19. References

[1] N. Chah, “Untangling deep learning from artificial intelligence and machine learning,” First Monday, vol. 24, no. 2, 2019. doi: 10.5210/fm.v24i2.8237. [Online]. Available: https://doi.org/10.5210/fm.v24i2.8237

[2] A. Vaswani et al., “Attention is all you need,” arXiv preprint arXiv:1706.03762, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[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,” arXiv preprint arXiv:2203.02155, 2022. [Online]. Available: https://arxiv.org/abs/2203.02155

[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,” arXiv preprint arXiv:2201.11903, 2022. [Online]. Available: https://arxiv.org/abs/2201.11903

[9] Z. Ji et al., “Survey of hallucination in natural language generation,” arXiv preprint arXiv:2202.03629, 2022. [Online]. Available: https://arxiv.org/abs/2202.03629

[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,” 2023. [Online]. Available: https://openai.com/research/gpt-4

[12] Google, “Gemini 3 Pro – Model Card,” 2024. [Online]. Available: https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf

[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," arXiv preprint arXiv:2310.01714, 2023. [Online]. Available: https://arxiv.org/abs/2310.01714

[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," arXiv preprint arXiv:2505.18059, 2025. [Online]. Available: https://arxiv.org/abs/2505.18059

[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, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto and P. Fung, "Survey of hallucination in natural language generation," arXiv preprint arXiv:2202.03629, 2022. [Online]. Available: https://arxiv.org/abs/2202.03629

[22] L. Ouyang et al., "Training language models to follow instructions with human feedback," arXiv preprint arXiv:2203.02155, 2022. [Online]. Available: https://arxiv.org/abs/2203.02155

[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] D. 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] 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] GitHub, “GitHub homepage,” Accessed: Dec. 1, 2025. [Online]. Available: https://github.com/

[35] Microsoft, “GitHub Copilot,” Accessed: Dec. 1, 2025. [Online]. Available: https://copilot.microsoft.com/

[36] Amazon, “CodeWhisperer documentation,” Accessed: Dec. 1, 2025. [Online]. Available: https://docs.aws.amazon.com/codewhisperer/

[37] Tabnine, “Tabnine AI assistant,” Accessed: Dec. 1, 2025. [Online]. Available: https://www.tabnine.com/

[38] Mintlify, “Mintlify Docs,” Accessed: Dec. 1, 2025. [Online]. Available: https://www.mintlify.com/

[39] Sourcery, “Sourcery AI,” Accessed: Dec. 1, 2025. [Online]. Available: https://www.sourcery.ai/

[40] Microsoft, “Copilot Chat,” Accessed: Dec. 1, 2025. [Online]. Available: https://copilot.cloud.microsoft/

[41] Replit, “Replit AI,” Accessed: Dec. 1, 2025. [Online]. Available: https://replit.com/ai

[42] JetBrains, “JetBrains AI assistant,” Accessed: Dec. 1, 2025. [Online]. Available: https://www.jetbrains.com/

[43] Semgrep, “Semgrep documentation,” Accessed: Dec. 1, 2025. [Online]. Available: https://semgrep.dev/

[44] Snyk, “Snyk Code,” Accessed: Dec. 1, 2025. [Online]. Available: https://snyk.io/product/snyk-code/

[45] Diffblue, “Diffblue Cover,” Accessed: Dec. 1, 2025. [Online]. Available: https://www.diffblue.com/

[46] Sourcegraph, “Cody AI,” Accessed: Dec. 1, 2025. [Online]. Available: https://sourcegraph.com/

[47] IBM, “Watsonx Code Assistant,” Accessed: Dec. 1, 2025. [Online]. Available: https://www.ibm.com/products/watsonx-code-assistant

[48] OpenAI, “Codex,” Accessed: Dec. 1, 2025. [Online]. Available: https://openai.com/

[49] Eclipse Foundation, “Repairnator,” Accessed: Dec. 1, 2025. [Online]. Available: https://projects.eclipse.org/

[50] NVIDIA, “NVIDIA Isaac SDK,” Accessed: Dec. 1, 2025. [Online]. Available: https://developer.nvidia.com/isaac

[51] DevRev, “DevRev Careers,” Accessed: Dec. 1, 2025. [Online]. Available: https://devrev.ai/

[52] Skydio, “Skydio Careers,” Accessed: Dec. 1, 2025. [Online]. Available: https://www.skydio.com/

[53] Boston Dynamics, “Boston Dynamics Homepage,” Accessed: Dec. 1, 2025. [Online]. Available: https://bostondynamics.com/

[54] Y. Zhao, “Exploration of computer programming teaching reform based on large language models” 2024. [Online]. Available: https://www.researchgate.net/publication/394653827_Exploration_of_Computer_Programming_Teaching_Reform_Based_on_Large_Language_Models

[55] T. Wang and N. Zhou, “Enhancing computer programming education with LLMs: A study on effective prompt engineering,” arXiv preprint arXiv:2407.05437, 2024. [Online]. Available: https://arxiv.org/abs/2407.05437

[56] 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

[57] 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

[58] 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

[59] N. Alfirević, M. Hell, and D. Rendulić, “Pedagogical qualities of artificial intelligence-assisted teaching: An exploratory analysis of a personal tutor in a voluntary business higher-education course,” *Applied Sciences*, vol. 15, no. 15, 8764, 2025. https://doi.org/10.3390/app15158764

[60] 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.

[61] 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.