JOURNAL OF SCIENCE
OF LAC HONG UNIVERSITY
ISSN: 2525 - 2186
Tạp chí Khoa học Lạc Hồng, **2025**, 22, 007-012

NGHIÊN CỨU KHẢ THI VÀ HIỆU NĂNG LLM TRÊN THIẾT BỊ DI ĐỘNG CHO HỖ TRỢ HỌC LẬP TRÌNH C++

Hà Hoàng Phúc[1], Nguyễn Tam Mạnh[1], Trương Hoàng Mẫn[1], Phạm Hoàng Phương[1], Võ Thị Ánh Nhi[1], Nguyễn Lê Vân Thanh[2*], Cao Thái Phương Thanh[1]
[1]*Trường Đại học Sài Gòn, Thành phố Hồ Chí Minh, Việt Nam*
[2]*Trường Đại học Sư Phạm Kỹ Thuật, Thành phố Hồ Chí Minh, Việt Nam*
* Tác giả liên hệ: *thanhnlv@hcmute.edu.vn*

THÔNG TIN BÀI BÁO

Ngày nhận: 07/5/2025
Ngày hoàn thiện: 11/7/2025
Ngày chấp nhận: 12/7/2025
Ngày đăng: 15/9/2025

TỪ KHÓA

Lập trình C++;
Thiết bị di động;
Lượng tử hóa;
Học lập trình ngoại tuyến;
DeepSeek.

TÓM TẮT

Học lập trình C++ là một quá trình phức tạp, đòi hỏi người học nắm vững cả cú pháp lẫn tư duy thuật toán. Nghiên cứu này nhằm đánh giá khả năng triển khai mô hình ngôn ngữ lớn (LLM) trên thiết bị di động để hỗ trợ quá trình học lập trình C++ hiệu quả hơn. Các mô hình được thử nghiệm gồm DeepSeek-Coder, Llama và Gemma, với các kỹ thuật tối ưu như lượng tử hóa (quantization) 4-bit và 8-bit nhằm giảm mức tiêu thụ tài nguyên. Thí nghiệm được thực hiện để đo độ chính xác khi giải bài tập C++, mức sử dụng bộ nhớ (VRAM, RAM) và tốc độ suy luận ở các mức tối ưu khác nhau. Kết quả cho thấy DeepSeek-Coder-1.3B giải đúng khoảng 40% bài, tiêu tốn 3.2GB VRAM – phù hợp cho điện thoại. Trong khi đó, DeepSeek-V2-Lite-Instruct (4-bit) đạt độ chính xác 64% nhưng yêu cầu 6GB VRAM, thích hợp hơn với laptop. Sau lượng tử hóa, mô hình có thể hoạt động ổn định trên thiết bị như Samsung A52S (RAM 8GB), với mức sử dụng RAM hệ thống khoảng 1.9GB – không bao gồm phần dùng cho hệ điều hành. Kết luận, việc triển khai LLM trên thiết bị di động là hoàn toàn khả thi và đầy tiềm năng trong việc hỗ trợ học lập trình. Nhóm nghiên cứu sẽ tiếp tục cải tiến hiệu năng và giao diện người dùng trong các bước tiếp theo.

FEASIBILITY AND PERFORMANCE STUDY OF LLMS ON MOBILE DEVICES FOR SUPPORTING C++ PROGRAMMING LEARNING

Ha Hoang Phuc[1], Nguyen Tam Manh[1], Truong Hoang Man[1], Pham Hoang Phuong[1], Vo Thi Anh Nhi[1], Nguyen Le Van Thanh[2*], Cao Thai Phuong Thanh[1]
[1]*Saigon University, Ho Chi Minh City, Vietnam*
[2]*HCMC University of Technology and Education, Ho Chi Minh City, Vietnam*
*Corresponding Author: *thanhnlv@hcmute.edu.vn*

THÔNG TIN BÀI BÁO

Received: May 7th, 2025
Revised: Jul 11st, 2025
Accepted: Jul 12nd, 2025
Published: Sep 15th, 2025

KEYWORDS

Programming in C++;
Mobile devices;
Quantization;
Learn offline programming;
DeepSeek.

ABSTRACT

Learning C++ programming is a complex process that requires mastering both syntax and algorithmic thinking. This study aims to evaluate the feasibility of deploying large language models (LLMs) on mobile devices to support users in learning C++ more effectively. The research involved testing models such as DeepSeek-Coder, Llama, and Gemma, and applying optimization techniques like 4-bit and 8-bit quantization to reduce hardware resource consumption. Experiments measured model accuracy on C++ tasks, memory usage (VRAM, RAM), and inference speed under different optimization levels. Results showed that DeepSeek-Coder-1.3B achieved the highest accuracy among mobile-friendly models, solving around 40% of C++ problems with 3.2GB of VRAM—suitable for smartphones. Meanwhile, DeepSeek-V2-Lite-Instruct (4-bit) reached 64% accuracy but consumed 6GB VRAM, making it more appropriate for laptops. After quantization, the model ran stably on devices such as the Samsung A52S (8GB RAM), requiring approximately 1.9GB of system RAM (excluding OS usage), which ensures acceptable performance on mid-range mobile devices. The findings confirm that deploying LLMs on mobile platforms is feasible and holds significant potential in supporting programming education. In the future, the research team will continue to optimize performance and improve the user interface to enhance the overall learning experience.

Doi: https://doi.org/10.61591/jslhu.22.789
Available online at: https://js.lhu.edu.vn/index.php/lachong

1. INTRODUCTION

In the rapidly advancing era of technology, programming has become an indispensable skill for information technology students. However, learning programming – especially with the C++ language – is a significant challenge for many students, particularly in the initial stages of learning. The main difficulties stem from the need to master syntax, algorithmic thinking, and the ability to analyze and solvelearning. Theith the rapid development of artificial intelligence, large language models (LLM) have proven their potential to effectively support the process of learning programming.Many recent studies, such as Khan et al. (2023) and Finnie-Ansley et al. (2022), indicate that models like Codex and ChatGPT can generate syntactically correct code, assist in debugging, and provide algorithm explanations, thereby helping to improve learning outcomes for programming students.

In addition, the trend of deploying LLMs on mobile devices is also attracting significant interest in the research community, with projects like LLaMA.cpp, MLC-LLM, and GGML demonstrating the potential to run compact models directly on mainstream hardware (Jiang et al., 2023; Deng et al., 2024).However, these studies mainly focus on model optimization techniques and have not deeply evaluated the applicability in education, especially in supporting programming learning through offline LLMs.

Current platforms supporting programming learning, such as W3Schools, Codecademy, or LeetCode, mainly require continuous Internet connectivity and provide limited targeted support for the C++ language. Moreover, most current AI tools (such as ChatGPT, Copilot) require a network connection and substantial hardware resources, making them difficult for students with limited learning conditions to access.

From there, the research gap is identified: there is currently no solution that implements compact, offline LLMs on mobile devices to support learning C++ programming, while also providing a comprehensive evaluation of performance, accuracy, and practical feasibility.

Therefore, the topic "Feasibility and Performance Study of LLM on Mobile Devices for Supporting C++ Programming Learning" was born with the aim of realizing the deployment of LLM models directly on smartphones, enabling students to learn programming anytime, anywhere.The project focuses on open-source models, small in size, optimized to run offline.Through this, the research team aims to achieve goals such as: Evaluating the performance and accuracy of LLM after quantization when running on mobile devices;Compare different models (DeepSeek, LLaMA, Gemma...) with different quantization formats (4-bit, 8-bit); Analyze the code generation capabilities and common errors of the models; Propose a direction for developing an offline LLM-integrated programming learning application for students.

The novel contributions of the topic include: (1) A comprehensive evaluation of LLM models quantized on mobile devices for the C++ code generation task; (2) Analysis of code generation errors, resource performance, and execution time, thereby determining the feasibility of application in resource-constrained learning environments; (3) Suggesting the design of an offline AI-supported learning system, opening a new direction for personalized education without the need for network connectivity.

2. CONTENT

2.1 Research methods

Dataset Preparation

The research uses the HumanEval-X (C++) dataset from HuggingFace, which is used as the main benchmark to evaluate large language models. This is a dataset consisting of 164 diverse C++ programming problems, covering basic programming concepts focused on foundational knowledge such as basic syntax, conditional structures, loops, functions, and arrays.

Some notable statistics from 100 random problems:

Arrays: 61 problems (61.0%).

If_Else: 90 problems (90.0%).

Loops: 90 problems (90.0%).

Some notable statistics from the total number of problems:

Loops: 146 problems (90.1%).

If_Else: 140 problems (86.4%).

Arrays: 99 problems (61.1%)

Each problem includes a problem description, function declaration, and unit test set (input and output), but does not contain a solution – suitable for evaluating the model's ability to generate code rather than for training. HumanEval-X was chosen because of its high representativeness to basic C++ programming skills and its extensibility to check correctness through automated testing.

Model Training and Optimization

In this study, the authors did not retrain or fine-tune the large language models from scratch. Instead, the Gemma, LLaMA, and DeepSeek models were selected from pre-trained open-source models. This selection is based on criteria such as small size (from 1 to under 30 billion parameters), compatibility with the GGUF format, popularity within the open-source community, and the ability to deploy on common devices. These models are all designed to generate source code for programming tasks, especially for short and clearly structured tasks as found in the HumanEval-X (C++) dataset.

After selecting the model, the team proceeded with the optimization process through quantization to reduce hardware resource requirements while still ensuring reasonable processing performance. This process involves converting the model's weights from high-precision floating-point format (FP32) to lower-precision integer formats such as INT8 or INT4, significantly reducing the model size and processing speed. Two quantization methods are applied, including:

Symmetric Quantization: All weight values are scaled to a value range symmetric around 0, suitable for balanced weight distributions.

Asymmetric Quantization: Weights are scaled in a skewed value domain, reducing information loss with asymmetric skewed distributions.

To perform the quantization process and deploy the model, the team uses the llama.cpp library – an open-source framework that allows running LLM directly on common CPUs and GPUs. Llama.cpp supports the GGUF (GPT-Generated Unified Format) – helping to reduce model size and optimize operation in resource-limited environments, typically mobile devices.

The GGUF quantization methods used

The specific quantization formats used in the study include: (1) Q4_K_M: Applied to 4-bit models such as DeepSeek-Coder-1.3B-it and LLaMA-3.2-3B-it-4bit; (2) Q8_0: Used for 8-bit models, such as LLaMA-3.2-3B-it-8bit; (3) Q5_K_M: Used in internal testing but not selected for reporting because it is not as optimized as Q4_K_M in mobile environments.

Especially, llama.cpp supports the K-Quant technique, a modern quantization method that divides the model into blocks and super-blocks, allowing for precise layer-wise quantization. K-Quant uses a selective strategy and weight rounding based on the scale factor to optimize the balance between model size and inference accuracy.

After optimization, the models are deployed in a testing environment and evaluated using the HumanEval-X (C++) dataset, with 100 randomly selected problems. Each problem is input into the model as a prompt, and the generated code is tested using a predefined unit test set to determine the metrics: accuracy, inference time, and resource consumption (RAM/VRAM). The evaluation process helps determine the model's performance on both personal computers and mobile devices, thereby assessing its feasibility for real-world deployment.

2.2 Results and Evaluation

Based on the implementation techniques in the previous section, the study evaluates the performance of multiple LLM models with scales from 1 to 27 billion parameters, to determine the feasibility of deployment on general-purpose hardware. The experiments are performed in a standardized environment with Docker, using Linux operating system, C++17 compiler, Python 11 and CUDA 12.5. The hardware configuration includes Intel Core i5-12400F CPU, 32GB RAM and Nvidia RTX 4070 Super 12GB GPU for desktop; and Samsung A52S phone (Snapdragon 778G 5G, 8GB RAM) running Android 14 for mobile testing.

The test dataset includes 100 C++ problems randomly selected from the HumanEval-X set with fixed seeds, ensuring objectivity. For each problem, the model is asked to generate code from the prompt, and the results are checked through predefined unit test scripts.

Evaluation of models under 8 billion parameters

The evaluated models include Gemma-3-1B-it, Llama-3.2-3B-it, DeepSeek-Coder-1.3B-it and their quantized versions (4-bit, 8-bit). For small models, the results are as follows:

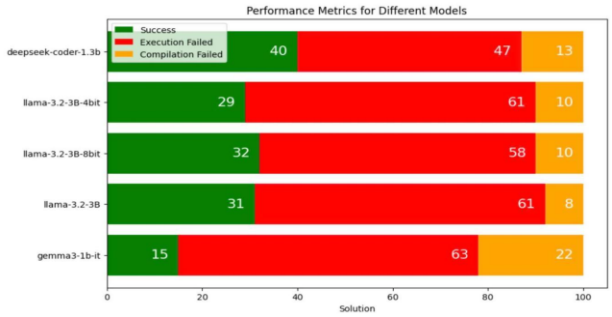

Figure 1. *State chart of the number of problems of models under 8 billion parameters*

- Regarding the number of successes: The DeepSeek-Coder-1.3B-it model achieved outstanding performance with 40 out of 100 problems solved correctly, while Gemma-3-1B-it only achieved 15 out of 100. The LLaMA 3.2-3B models have average performance, ranging from 29 to 32 successful problems. This can be explained by the fact that DeepSeek-Coder-1.3B-it was specially trained for programming tasks (code-specialized tuning), which helps increase accuracy and the ability to generate valid code higher than general language models like Gemma or LLaMA.
- Regarding the number of failures due to not passing the unit test (Execution Failed): DeepSeek-Coder-1.3B-it had the lowest number of failures (47 times), showing a high ability to generate approximate code. Meanwhile, Gemma-3-1B-it had the highest failure rate (63 times). The LLaMA 3.2-3B models had average performance, ranging from 58 to 61 failures.
- Regarding the number of failures due to code compilation (Compilation Failed): DeepSeek-Coder-1.3B-it also shows stability with only 13 code compilation failures, compared to 22 code compilation failures in Gemma-3-1B-it. The LLaMA 3.2-3B models have average performance, ranging from 8 to 10 failures.

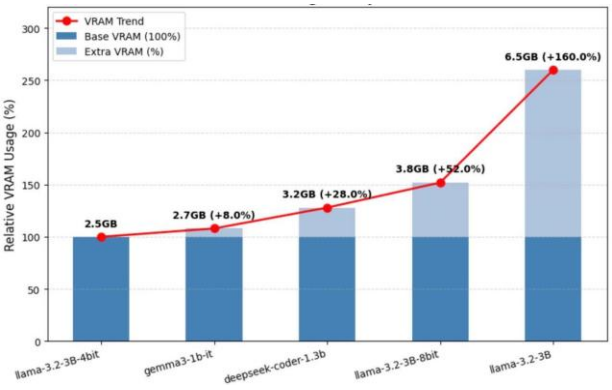

Figure 2. *Capacity chart of models under 8 billion parameters*

- Regarding resource usage: Llama-3.2-3B-it-4bit uses the least VRAM (2.5GB), while the non-quantized version of Llama-3.2-3B exceeds the common 6GB

VRAM limit on mid-range laptops (6.5GB). Although it uses less VRAM, models like Llama-3.2-3B-it-4bit do not achieve the same high performance as DeepSeek, indicating that quantization needs to be combined with good pretraining to maintain accuracy. Additionally, the Deepseek-Coder-1.3b model can occupy 50% of the user's computer's VRAM capacity. (3.2GB). The Gemma-3-1b-it model has the second lowest VRAM usage, only after the Llama-3.2-3B-bit-4bit.

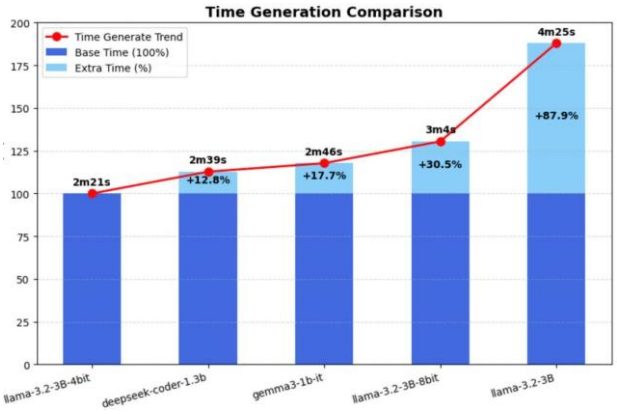

Figure 3. *Execution time graph of models under 8 billion parameters.*

- Regarding inference speed: Quantization significantly reduces execution time. Llama-3.2-3B-it-4bit has the fastest processing speed. The DeepSeek-Coder-1.3B-it model has 37.93% higher accuracy than Llama-3.2-3B-4bit but only takes 12.8% more inference time.

Evaluating models over 8 billion parameters

The models in this group include Llama-3-8B-it, Gemma-3-12B-it, DeepSeek-Coder-V2-Lite-Instruct and Gemma-3-27B-it. To ensure the ability to deploy on common hardware, the models are quantized down to 4-bit or 8-bit. The results show:

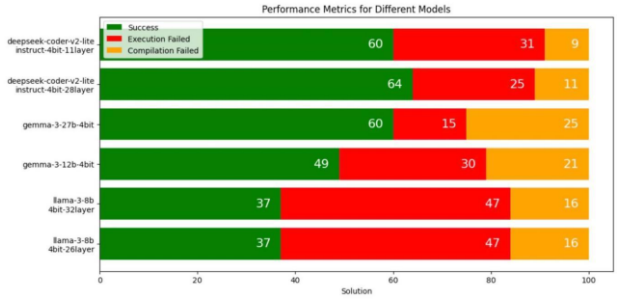

Figure 4. *State diagram of the number of problems for models with 8 billion parameters or more*

- Accuracy: DeepSeek-Coder-V2-Lite-Instruct-4bit achieved the highest performance with an accuracy rate of up to 64%, surpassing Llama-3-8B-it-4bit by 73%. The superiority of DeepSeek-Coder-V2-Lite-Instruct is largely due to the distillation technique from larger models like CodeLlama-34B, which helps the model retain important knowledge while reducing the number of parameters. Compared to recent reports on the Open LLM Leaderboard, this model performs at least as well as or better than StarCoder-15B on some programming tasks, despite being significantly smaller in size.

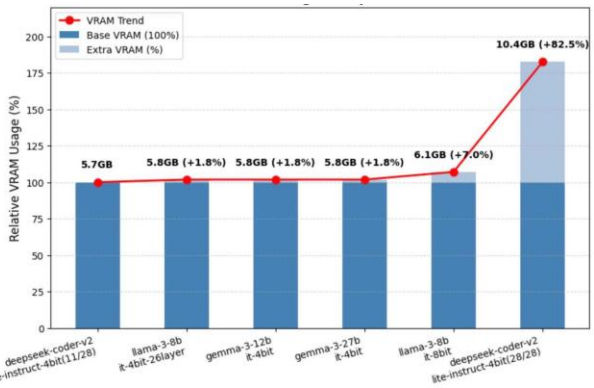

Figure 5. *Capacity diagram of models with 8 billion parameters or more.*

- Storage capacity and architecture: Thanks to the GGUF format, the model can be split: part runs on the GPU, the rest on the RAM. Choosing 11 layers to load onto the GPU based on the 6GB VRAM limit shows a balance between speed and memory, but this number has not yet proven to be globally optimal. Experiments with different numbers of layers (e.g., 8, 13, 15) could help determine the optimal performance threshold.

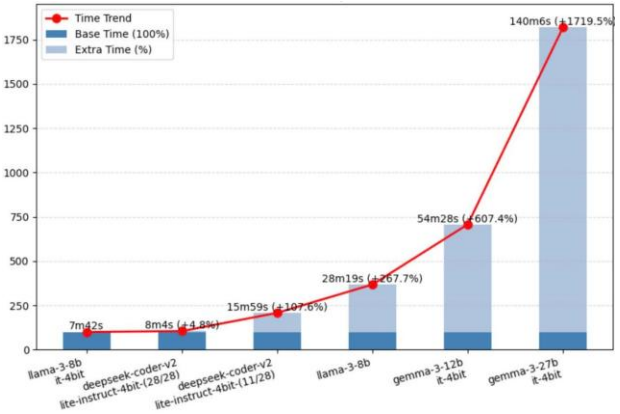

Figure 6. *Execution time chart of models with 8 billion parameters or more*

- Inference speed: The combination of GPU and CPU computation increases inference time due to task switching latency. However, DeepSeek-Coder-V2-Lite-Instruct-4bit still maintains reasonable processing speed and high accuracy, demonstrating the potential of the hybrid architecture (CPU + GPU) in non-specialized environments. However, DeepSeek-Coder-V2-Lite-4bit still maintains a good speed, completing about 60% of the total number of problems.

General summary:

The Deepseek-Coder-1.3b-it model will be the most suitable model to be able to run directly on mobile phones while still maintaining high accuracy.

The Deepseek-Coder-V2-Lite-Instruct-4bit model will be the suitable model to run on computers of ordinary users.

2.3 Limitations and development directions

Limitations:

The current study still has some notable limitations in both technical and methodological aspects.

First, the errors from the collected logs indicate two main groups of errors:

1. Compilation Failed: Accounts for a significant proportion, including common errors such as:

- Missing #include standard library (<stack>, <vector>, <string>, etc.)
- Not using std:: or missing using namespace std;
- Scope error (using undeclared variables)
- Syntax error (;, {}, ()) and data type error.

2. Execution Failed: Includes errors such as:

- Assertion Failed: The code runs but returns incorrect results compared to the test case. Algorithmic logic errors are the majority (e.g., false conditions, incomplete string processing, off-by-one…)
- Runtime Timeout: Some cases have no obvious errors but exceed the processing time limit.

Secondly, the research still has limitations in experimental design: (1) Small sample size: Only 100 problems from the HumanEval-X dataset were used, which is not enough to comprehensively represent C++ programming skills in practice; (2) The scope of model evaluation is still limited: The study only uses accuracy as the main criterion; there are no other supplementary indicators such as code comprehensibility, complexity, or pedagogical quality. (3) Limited testing devices: Only evaluated on one phone model (Samsung A52S) and one computer configuration (RTX 4070), so it does not fully reflect the model's deployment capabilities on other popular devices; (4) Lack of real user verification: The research has not conducted surveys of learners or analyzed the real experiences of end users; (5) Not compared with SOTA models: The results have not been placed in a broader context to compare with other modern models that may achieve higher accuracy or better optimization capabilities.

Development direction:

Based on the experimental results and the identified limitations, this study proposes several development directions to further enhance and expand the application in the future. Firstly, the research aims to develop a complete C++ programming learning application that is user-friendly and capable of running offline on devices with standard configurations. This application will integrate optimized LLMs and support features such as code generation, error correction suggestions, line-by-line explanations, automatic assessment, and practice exercises.

In parallel, the study focuses on improving the accuracy of code generation through fine-tuning on specialized datasets for the C++ language, as well as analyzing and addressing common errors such as compilation errors, logic errors, and runtime failures. Optimizing the quantization algorithms is also a key direction, involving the exploration and application of advanced techniques like K-Quant, Group-wise Quantization, or Mixed Precision to achieve a balance between processing efficiency and resource consumption. These optimizations will allow the models to perform better on mobile devices or low-end computers.

Additionally, the study aims to expand support to other popular programming languages such as Python, Java, or JavaScript, in order to serve a broader range of learners. Personalizing the learning process is also a key focus, with the development of a system to track user progress and analyze errors to provide tailored learning suggestions based on individual skill levels and learning needs. Finally, the study aims to enhance the natural interaction and communication capabilities between the learner and the model, allowing the model to act as a virtual tutor that can flexibly respond to questions. This interactive approach supports learners in developing a proactive and effective problem-solving mindset.

3. CONCLUSION

In the context of the increasing demand for programming learning and the strong development of artificial intelligence-based support tools, the topic "Feasibility and Performance Study of LLM on Mobile Devices for Supporting C++ Programming Learning" has demonstrated the feasibility of deploying large language models directly on mobile devices. Through practical experiments with the DeepSeek-Coder-1.3B model, the study shows that the model can operate stably, achieve significant accuracy and consume resources at a suitable level, thereby affirming the application potential of LLM in supporting offline programming learning.

Due to time and resource constraints, the study has not yet conducted statistical significance testing. This is an important direction that needs to be implemented in future studies to determine the reliability of the differences in results between the models. In addition, the study analyzed the effectiveness of the quantization technique in reducing the model size and increasing the processing speed. However, this process also leads to some limitations in accuracy. Tracking the compilation errors and execution errors generated by the model has provided practical insights into the AI's C++ code generation capabilities and also serves as a basis for proposing improvements to enhance the quality of learning support.

In the future, the research team aims to develop a dedicated application with a friendly interface, integrating step-by-step learning instructions, practice exercises, and progress tracking tools. Further research will focus on improving the accuracy of the generated code, optimizing the quantization technique to suit low-configuration devices, as well as expanding to other programming languages such as Python or Java. The results achieved from the project not only affirm the potential of LLM in programming education but also open up development directions for smart learning tools, contributing to improving the quality of information technology training in the future.

4 REFERENCES

[1] P. Ross, "Sistemas de tutoría inteligente," *Intelligent Systems Journal*, vol. 3, no. 4, pp. 194–203, **1987**.

[2] G.-J. Hwang, "Un modelo de mapa conceptual para el desarrollo de sistemas de tutoría inteligente," Computers & Education, vol. 40, no. 3, pp. 217–235, **2003**.

[3] T. Baker y L. Smith, "¿Educ-AI-ción reiniciada? Explorando el futuro de la inteligencia artificial en escuelas y colegios," Futuro de la Educación, **2019**.

[4] S. Bayne, "Teacherbot: Interventions in automated teaching," Teaching in Higher Education, vol. 20, no. 4, pp. 455–467, **2015**.

[5] "Google responds to DeepSeek with the ultra-lightweight AI model Gemma 3," Dantri, Mar. **2025**. [Online]. Available: https://dantri.com.vn/cong-nghe/google-dap-tra-deepseek-bang-mo-hinh-ai-sieu-nhe-gemma-3-20250313105241524.htm.

[6] "Optimizing large language models with LLaMACpp and running on mobile," AI Lab, **2024**. [Online]. Available: https://ailab.siu.edu.vn/article/49/toi-uu-mo-hinh-ngon-ngu-lon-voi-llamacpp-va-chay-tren-ien-thoai.

[7] "What is DeepSeek?" Growstack, **2024**. [En línea]. Disponible en: https://www.growstack.vn/deepseek-la-gi.

[8] Gemini Team, "Gemma: Open models based on Gemini research and technology," **2024**.

[9] Gemini Team, "Gemini: Una familia de modelos multimodales altamente capaces," **2023**.

[10] "Informe Técnico de Gemma 3," Google AI, **2024**.

[11] "Gemma 3 model overview," Google AI for Developers, **2024**. [En línea]. Disponible:

https://developers.google.com/gemma3.

[12] LLaMA Team, "The Llama 3 Herd of Models," AI @ Meta, **2024**.

[13] "DeepSeek-Coder: Cuando el Modelo de Lenguaje Grande se Encuentra con la Programación," El Auge de la Inteligencia de Código, **2024**.

[14] "Application of chatbots in the field of education," VmixGPT, **2024**. [Online]. Available: https://vmixgpt.com/ung-dung-chatbot-trong-linh-vuc-giao-duc.

[15] "Application of chatbots in training," OES, **2024**. [Online]. Available: https://oes.vn/ung-dung-chatbot-trong-dao-tao.

[16] "A visual guide to quantization," Maarten Grootendorst's Newsletter, **2024**. [En línea]. Disponible:

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization.

[17] "Llama.cpp pull request #1684," GitHub, **2024**. [En línea]. Disponible en: https://github.com/ggml-org/llama.cpp/pull/1684.