

INTEGRATING LARGE LANGUAGE MODELS WITH AUTOMATED ASSESSMENT SYSTEMS: A CHAIN-OF-THOUGHT APPROACH FOR PEDAGOGICAL FEEDBACK AND MEMORY OPTIMIZATION

Nguyen Tuan Linh

TNU - University of Information and Communication Technology

ARTICLE INFO		ABSTRACT
Received:	29/5/2026	The evolution of smart education has established Automated Grading Systems as essential tools in programming instruction. However, the majority of current platforms merely yield binary pass/fail verdicts, failing to provide formative pedagogical feedback to help learners identify underlying logical flaws. While Large Language Models demonstrate substantial potential in source code analysis, their direct application via zero-shot prompting remains highly susceptible to hallucinations. This results in inconsistent evaluations, particularly under stringent temporal and spatial constraints typical in competitive programming. To address these critical limitations, this study proposes a multi-tier modular architecture integrating Large Language Models with Chain-of-Thought reasoning. The proposed framework directs the artificial intelligence through a rigorous sequential reasoning pipeline encompassing semantic decoding, algorithmic complexity evaluation, boundary fault detection, and memory optimization. Notably, it introduces an implicit rule module for the proactive suggestion of <code>__int128</code> data types to mitigate integer overflow risks. Empirical evaluations across 1,250 C/C++ submissions reveal that the Chain-of-Thought architecture significantly outperforms the zero-shot baseline. The system achieves an 8.7% improvement in Pass@1 accuracy, a 9.2% enhancement in algorithmic complexity analysis, a 22.2% increase in IndexError detection rates, and a 57.6% precision surge in <code>__int128</code> recommendations.
Revised:	29/6/2026	
Published:	30/6/2026	
KEYWORDS		
Large language models		
Automated grading systems		
Chain-of-thought		
Pedagogical feedback		
Memory optimization		

TÍCH HỢP MÔ HÌNH NGÔN NGỮ LỚN VÀO HỆ THỐNG CHẤM ĐIỂM LẬP TRÌNH TỰ ĐỘNG: TIẾP CẬN CHUỖI SUY LUẬN CHO PHẢN HỒI SỰ PHẠM VÀ TỐI ƯU BỘ NHỚ

Nguyễn Tuấn Linh

Trường Đại học Công nghệ Thông tin và Truyền thông – ĐH Thái Nguyên

THÔNG TIN BÀI BÁO	TÓM TẮT
Ngày nhận bài: 29/5/2026	Sự phát triển của giáo dục thông minh đã thúc đẩy việc ứng dụng các Hệ thống chấm điểm tự động trong đào tạo lập trình. Tuy nhiên, đa số nền tảng hiện nay chủ yếu trả về kết quả đúng/sai mà chưa cung cấp phản hồi mang tính sư phạm để hỗ trợ người học cải thiện tư duy thuật toán. Trong khi đó, các Mô hình ngôn ngữ lớn cho thấy tiềm năng đáng kể trong phân tích mã nguồn, nhưng khi áp dụng trực tiếp theo cơ chế zero-shot thường xuất hiện hiện tượng “ảo giác” và thiếu nhất quán trong các bài toán có ràng buộc tài nguyên nghiêm ngặt. Nghiên cứu này đề xuất kiến trúc mô-đun đa tầng kết hợp mô hình ngôn ngữ lớn với kỹ thuật Chuỗi suy luận dành cho lập trình thi đấu. Hệ thống thực hiện quy trình suy luận tuần tự gồm phân tích ngữ nghĩa chương trình, đánh giá độ phức tạp thuật toán, phát hiện lỗi biên như IndexError và tối ưu bộ nhớ. Đặc biệt, mô hình có khả năng chủ động đề xuất kiểu dữ liệu <code>__int128</code> nhằm hạn chế tràn số nguyên. Thử nghiệm trên 1.250 bài nộp C/C++ cho thấy phương pháp Chuỗi suy luận vượt trội hơn baseline zero-shot, cải thiện 8,7% độ chính xác Pass@1, tăng 9,2% năng lực phân tích thuật toán, nâng tỷ lệ phát hiện IndexError thêm 22,2% và cải thiện 57,6% độ chính xác khi sử dụng <code>int128</code> .
Ngày hoàn thiện: 29/6/2026	
Ngày đăng: 30/6/2026	
TỪ KHÓA	
Mô hình ngôn ngữ lớn	
Hệ thống chấm điểm tự động	
Chuỗi suy luận	
Phản hồi sư phạm	
Tối ưu bộ nhớ	

DOI: <https://doi.org/10.34238/tnu-jst.15929>

Email: ntlinh.khmt@ictu.edu.vn

<http://jst.tnu.edu.vn>

402

Email: jst@tnu.edu.vn

1. Giới thiệu

Sự phát triển của giáo dục thông minh và các nền tảng học trực tuyến mở (MOOCs) đang làm thay đổi đáng kể phương pháp giảng dạy khoa học máy tính. Trong bối cảnh đó, các Hệ thống chấm điểm tự động (Automatic Grading Systems - AGS) và nền tảng Online Judge (OJ) ngày càng được sử dụng rộng rãi nhờ khả năng xử lý khối lượng lớn bài nộp với tốc độ và độ chính xác cao [1], [2]. Từ đầu những năm 2000, các hệ thống như Web-CAT và Kattis đã trở thành công cụ phổ biến trong đào tạo lập trình [3]. Theo Jung và đồng sự [1], phần lớn các OJ vận hành dựa trên cơ chế thực thi động kết hợp kiểm thử hộp đen trong môi trường sandbox nhằm đảm bảo tính khách quan và khả năng mở rộng.

Tuy nhiên, các hệ thống OJ truyền thống vẫn còn nhiều hạn chế về mặt sự phạm. Phần lớn nền tảng hiện nay chỉ cung cấp các phản hồi ngắn như “Accepted”, “Wrong Answer” hoặc “Time Limit Exceeded”, chưa đủ để hỗ trợ người học hiểu nguyên nhân lỗi và cải thiện thuật toán [3]. Richards và đồng sự [4] cho rằng sinh viên thường gặp khó khăn khi xử lý các trường hợp biên (edge cases). Ngoài ra, các hệ thống hiện tại hầu như chưa đánh giá được chất lượng thiết kế chương trình; những đoạn mã có độ phức tạp chu trình (cyclomatic) cao vẫn có thể đạt điểm tối đa nếu vượt qua toàn bộ test-case [5]. Ngay cả khi tích hợp các công cụ phân tích tĩnh như linting, hệ thống vẫn dễ tạo ra cảnh báo giả và thiếu linh hoạt trong môi trường lập trình thực tế [6].

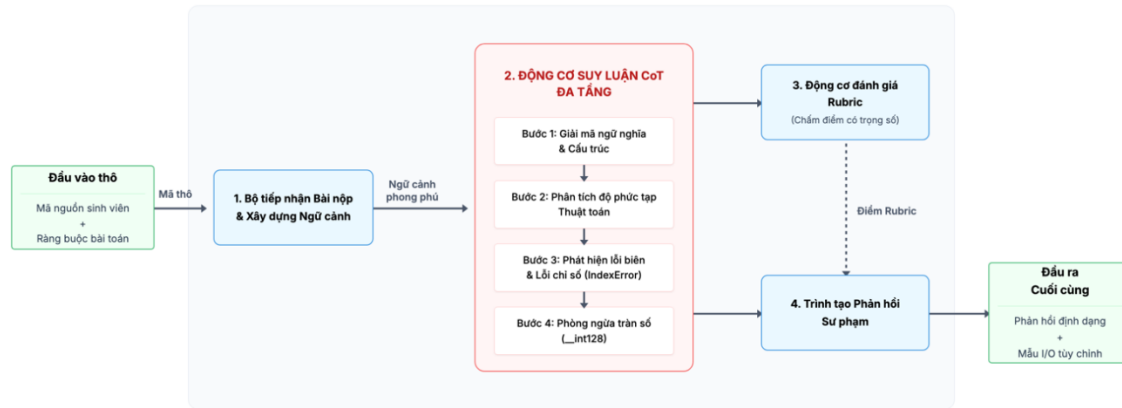
Sự xuất hiện của các Mô hình ngôn ngữ lớn (LLM) như GPT, Llama và DeepSeek đã mở ra nhiều tiềm năng cho đánh giá mã nguồn tự động [6]. Khác với các phương pháp học máy truyền thống, LLM có khả năng xử lý trực tiếp mã nguồn dưới dạng ngôn ngữ tự nhiên, từ đó hỗ trợ phân tích ngữ nghĩa chương trình, suy luận luồng điều khiển và sinh phản hồi chi tiết [7]-[9]. Nhiều nghiên cứu đã chứng minh hiệu quả của hướng tiếp cận này. Chẳng hạn, CodEv sử dụng cơ chế bỏ phiếu để giảm hiện tượng “ảo giác” [10], StepGrade khai thác GPT-4 nhằm đánh giá cả tính đúng đắn lẫn mức độ tối ưu của thuật toán [11], trong khi A-ProS có khả năng tự động sửa lỗi trên các bài toán Codeforces [12]. Những kết quả này cho thấy LLM có thể trở thành công cụ hỗ trợ học tập hiệu quả [13]. Tuy nhiên, phương pháp này vẫn tồn tại nhiều hạn chế như phản hồi thiếu nhất quán, khó xử lý các ràng buộc nghiêm ngặt về thời gian và bộ nhớ, đồng thời dễ bị đánh lừa bởi các chương trình đúng cú pháp nhưng sai logic [11]. Bên cạnh đó, để hạn chế gian lận học thuật, AI cần được thiết kế theo hướng cung cấp gợi ý thay vì đưa ra lời giải hoàn chỉnh [14].

Để cải thiện khả năng suy luận của LLM, kỹ thuật Chuỗi suy luận (Chain-of-Thought - CoT) được xem là một hướng tiếp cận triển vọng [15]. Thay vì đưa ra kết quả trực tiếp, CoT hướng mô hình phân tách bài toán thành nhiều bước suy luận trung gian nhằm kiểm tra logic trước khi đưa ra đánh giá cuối cùng [15]. Các nghiên cứu như SCoT và CGO cho thấy CoT giúp giảm lỗi logic, cải thiện từ 5% đến 12% tỷ lệ Pass@1 và sử dụng tài nguyên hiệu quả hơn so với nhiều kỹ thuật phản xạ khác [10], [16]. Khi tích hợp vào các hệ thống như StepGrade, CoT cho phép mô hình đánh giá đồng thời cấu trúc dữ liệu, thuật toán và chất lượng thiết kế chương trình [11].

Trong môi trường lập trình thi đấu như Olympic Tin học hay ACM-ICPC, các bài toán thường đi kèm những ràng buộc nghiêm ngặt về hiệu năng và quản lý bộ nhớ. Một trong những lỗi phổ biến là tràn số nguyên (Integer Overflow). Dù kiểu long long 64-bit hỗ trợ giá trị tới khoảng 9×10^{18} , nhiều bài toán tổ hợp hoặc phân tích thừa số nguyên tố vẫn có thể sinh ra giá trị trung gian vượt ngưỡng này [14], [17]. Vì vậy, cộng đồng lập trình thi đấu thường sử dụng kiểu `__int128` của GCC/Clang [18], dù kiểu dữ liệu này yêu cầu người học tự xây dựng cơ chế nhập/xuất riêng [19]. Đây là kỹ năng mà nhiều hệ thống AGS tích hợp AI hiện nay vẫn chưa hỗ trợ hiệu quả [16]. Bên cạnh đó, lỗi truy cập vượt quá giới hạn của mảng (IndexError) cũng là nguyên nhân phổ biến gây ra lỗi Undefined Behavior hoặc Segmentation Faults trong C/C++ [20]. Mặc dù một số nghiên cứu như CODERANKER đã thử nghiệm dự đoán lỗi runtime bằng mạng nơ-ron [21], việc kết hợp CoT để mô phỏng trạng thái bộ nhớ và phân tích ranh giới vòng lặp vẫn chưa được khai thác đầy đủ.

Xuất phát từ những khoảng trống trên, nghiên cứu đề xuất một kiến trúc đánh giá mã nguồn đa tầng kết hợp giữa LLM và kỹ thuật CoT dành cho môi trường lập trình thi đấu. Khác với các

hệ thống AGS truyền thống chỉ tập trung vào kết quả đầu ra, kiến trúc đề xuất hướng tới phân tích toàn diện cấu trúc và logic chương trình thông qua các bước suy luận tuần tự như giải mã ngữ nghĩa, đánh giá độ phức tạp thuật toán, phát hiện lỗi IndexError và hỗ trợ xử lý tràn số bằng `__int128`. Qua đó, hệ thống không chỉ đóng vai trò chấm điểm tự động mà còn trở thành công cụ hỗ trợ học tập có khả năng giải thích và định hướng hiệu quả hơn cho người học.



Hình 1. Kiến trúc tổng thể của hệ thống Chấm điểm Tự động (AGS) đề xuất. Sơ đồ minh họa quá trình chuyển đổi mã nguồn thô và các ràng buộc bài toán thành phản hồi định dạng mang tính sự phạm thông qua Động cơ suy luận Chuỗi suy luận (CoT) đa tầng và mô-đun Đánh giá Rubric có trọng số

2. Kiến trúc đề xuất

Nhằm đáp ứng nhu cầu thực tiễn trong giảng dạy lập trình và khai thác hiệu quả tiềm năng của AI tạo sinh, nghiên cứu đề xuất một kiến trúc Đánh giá Mã nguồn Tự động dựa trên kỹ thuật Chuỗi suy luận (Chain-of-Thought - CoT). Thay vì thay thế hoàn toàn cơ chế chấm điểm truyền thống dựa trên sandbox, hệ thống được xây dựng như một tầng nhận thức bổ sung, có khả năng phân tích mã nguồn theo nhiều bước suy luận liên tiếp để tạo ra các phản hồi mang tính giải thích và hỗ trợ học tập sâu hơn. Cách tiếp cận này không chỉ giúp đánh giá tính đúng đắn của chương trình mà còn hỗ trợ phát hiện lỗi logic, phân tích độ phức tạp thuật toán và đưa ra các gợi ý tối ưu hóa phù hợp với môi trường lập trình thi đấu.

2.1. Mô hình Kiến trúc Tổng thể

Hệ thống được thiết kế theo kiến trúc mô-đun đa tầng (Hình 1), trong đó các thành phần chức năng phối hợp liên tục để chuyển đổi mã nguồn đầu vào thành các phản hồi đánh giá có tính phân tích và hỗ trợ học tập.

Bộ tiếp nhận và xây dựng ngữ cảnh: Khác với các nền tảng OJ truyền thống, mô-đun này không chỉ tiếp nhận mã nguồn mà còn xây dựng ngữ cảnh đầy đủ cho quá trình đánh giá, bao gồm mô tả bài toán, ràng buộc dữ liệu vào/ra và giới hạn tài nguyên hệ thống. Ngoài ra, các lỗi phát sinh trong quá trình thực thi động như TLE hay RTE cũng được thu thập nhằm cung cấp thêm tín hiệu hỗ trợ cho AI trong quá trình suy luận.

Động cơ suy luận CoT đa tầng: Đây là thành phần trung tâm sử dụng các mô hình ngôn ngữ lớn như GPT-4o hoặc Claude 3.5 để thực hiện phân tích mã nguồn. Thay vì đánh giá trực tiếp, mô hình tiến hành suy luận theo nhiều bước liên tiếp nhằm xử lý các vấn đề liên quan đến ngữ nghĩa chương trình, độ phức tạp thuật toán, quản lý bộ nhớ và nguy cơ tràn số nguyên.

Động cơ đánh giá Rubric: Thành phần này chuyển đổi các phân tích định tính từ mô hình CoT thành hệ thống tiêu chí đánh giá chuẩn hóa, qua đó lượng hóa kết quả thành điểm số cụ thể nhằm đảm bảo tính nhất quán và công bằng trong quá trình chấm điểm.

Bộ tạo phản hồi sự phạm: Dựa trên kết quả từ Rubric và luồng suy luận CoT, mô-đun này sinh ra báo cáo phản hồi bằng ngôn ngữ tự nhiên, bao gồm phân tích lỗi, giải thích nguyên nhân

và các gợi ý định hướng để hỗ trợ người học cải thiện chương trình. Đối với các bài toán xử lý số lớn, hệ thống còn cung cấp các khuôn mẫu nhập/xuất phù hợp cho kiểu dữ liệu `__int128`.

2.2. Thiết kế Kỹ thuật Prompt Chuỗi suy luận (CoT) trong đánh giá mã nguồn

Hiệu quả của các mô hình ngôn ngữ lớn phụ thuộc đáng kể vào cách thiết kế prompt và tổ chức quá trình suy luận. Trong nghiên cứu này, hệ thống áp dụng kỹ thuật Chuỗi suy luận có cấu trúc (Structured CoT) kết hợp thẻ `<thinking>` nhằm định hướng mô hình thực hiện phân tích theo từng bước rõ ràng, từ đó giảm thiểu hiện tượng “ảo giác” và nâng cao tính nhất quán trong đánh giá mã nguồn. Quy trình suy luận được xây dựng gồm bốn giai đoạn chính.

Bước 1: Phân tích ngữ nghĩa và cấu trúc chương trình

Mục tiêu của bước này là chuyển đổi mã nguồn C/C++ sang dạng mã giả (pseudocode) để mô hình hiểu đúng ý đồ thuật toán trước khi thực hiện đánh giá. Prompt được thiết kế nhằm buộc LLM phân tích luồng điều khiển, xác định các cấu trúc dữ liệu quan trọng như mảng, hashmap hoặc hàng đợi ưu tiên, đồng thời làm rõ vai trò của các vòng lặp và điều kiện rẽ nhánh trong chương trình.

Bước 2: Phân tích độ phức tạp thuật toán

Ở giai đoạn này, hệ thống đánh giá tài nguyên tính toán và khả năng bảo trì mã nguồn. Mô hình được yêu cầu ước lượng độ phức tạp Big-O và so sánh với giới hạn thời gian thực thi phổ biến trong lập trình thi đấu (khoảng 10^8 phép tính mỗi giây). Đồng thời, hệ thống tính toán độ phức tạp Cyclomatic (CC) dựa trên số lượng đường dẫn thực thi độc lập nhằm xác định mức độ rủi ro và khả năng mở rộng của chương trình.

Bước 3: Phát hiện lỗi biên và IndexError

Mục tiêu của bước này là hoạt động như một công cụ phân tích tĩnh nhằm phát hiện sớm các lỗi truy cập bộ nhớ. Thông qua kỹ thuật “dry-run”, mô hình mô phỏng quá trình thực thi của các vòng lặp và điều kiện dừng với kích thước mảng khai báo. Nếu phát hiện nguy cơ truy cập vượt biên, hệ thống sẽ sinh cảnh báo liên quan đến `IndexError` hoặc `Segmentation Faults`. Cách tiếp cận này đặc biệt hữu ích đối với các thuật toán xử lý chuỗi hoặc kỹ thuật hai con trỏ.

Bước 4: Tối ưu bộ nhớ và phát hiện nguy cơ tràn số

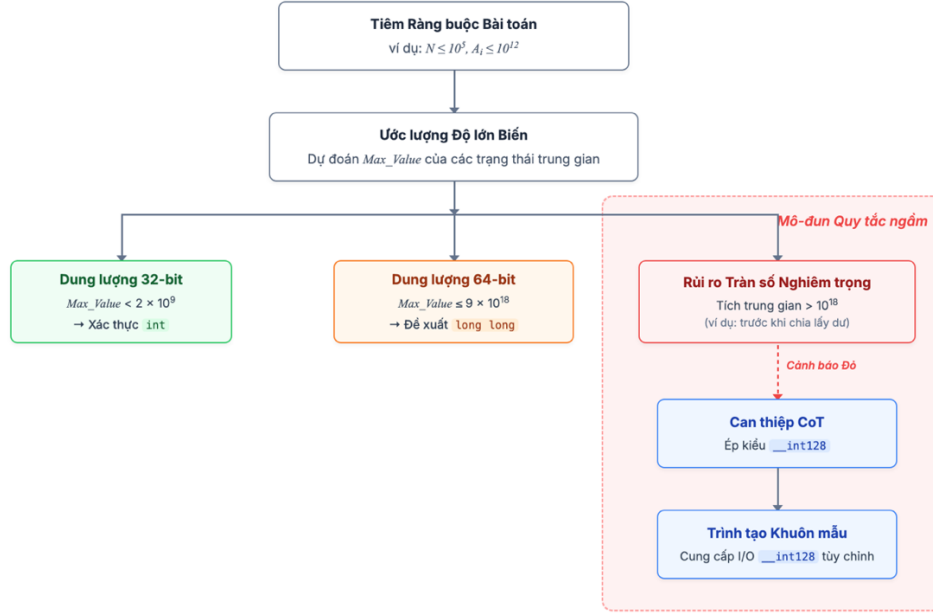
Ở bước cuối cùng, hệ thống tập trung đánh giá khả năng sử dụng bộ nhớ và rủi ro tràn số nguyên trong các phép tính trung gian. Các ràng buộc dữ liệu đầu vào như $N \leq 10^5$ hoặc $A_i \leq 10^{12}$ được tích hợp trực tiếp vào prompt để mô hình suy luận về phạm vi giá trị của từng biến trong quá trình thực thi. Nhờ đó, hệ thống có thể đưa ra cảnh báo và đề xuất sử dụng kiểu dữ liệu phù hợp như `__int128` đối với các bài toán xử lý số lớn (xem Hình 2).

Nhằm đảm bảo tính minh bạch và khả năng tái lập của nghiên cứu, cấu trúc prompt nền tảng được thiết kế và mô tả một cách rõ ràng. Kiến trúc này tách biệt giữa Thông điệp hệ thống (System Message), đóng vai trò định nghĩa nhiệm vụ và hành vi của mô hình, với Ngữ cảnh người dùng (User Context), nơi cung cấp dữ liệu động như đề bài, mã nguồn và các ràng buộc thực thi. Điểm quan trọng của phương pháp là mô hình phải thực hiện toàn bộ quá trình suy luận trung gian bên trong thẻ `<thinking>` trước khi tạo ra kết quả đánh giá cuối cùng theo Rubric chuẩn hóa. Cách tổ chức này giúp kiểm soát luồng suy luận của LLM, hạn chế hiện tượng “ảo giác” và nâng cao tính nhất quán trong quá trình đánh giá mã nguồn. Cấu trúc mã giả (pseudocode) của prompt được chuẩn hóa như sau:

```
{
  "system_message": "Expert Grader. Evaluate C/C++ via 4-step CoT (Semantic, Complexity, Boundary, Overflow). Use <thinking> tags.",
  "user_context": { "problem": "[...]", "constraints": "N<=10^5, A_i<=10^12", "code": "[...]" },
  "output_format": "<thinking>\n[Step 1 to 4 reasoning]\n</thinking>\n<json>\n{\n  \"C_func\": [\".\"],\n  \"C_comp\": [\".\"],\n  \"C_opt\": [\".\"],\n  \"C_style\": [\".\"],\n  \"feedback\": \"[...]\n\"}\n</json>"}
}
```

Thiết kế prompt này hoạt động như một cơ chế kiểm soát suy luận nhằm nâng cao độ tin cậy của quá trình đánh giá mã nguồn. Thông qua việc sử dụng thẻ `<thinking>` như một không gian suy luận trung gian, hệ thống buộc mô hình AI phải thực hiện phân tích tuần tự trước khi đưa ra

kết quả cuối cùng, từ đó giảm thiểu hiện tượng “ảo giác” thường gặp ở các phương pháp zero-shot (là cách yêu cầu mô hình thực hiện một nhiệm vụ mà không cung cấp ví dụ minh họa trước đó). Trong quá trình này, mô hình được yêu cầu rà soát đầy đủ các ràng buộc về bộ nhớ, giới hạn vòng lặp và tài nguyên thực thi nhằm đảm bảo các phản hồi sinh ra không chỉ đúng về mặt logic mà còn phù hợp với điều kiện thực tế của môi trường lập trình thi đấu. Nhờ đó, hệ thống có thể tạo ra các phản hồi sự phạm có tính giải thích và độ nhất quán cao hơn.



Hình 2. Cây quyết định của Mô-đun Quy tắc ngầm cho tối ưu hóa bộ nhớ. Sơ đồ minh họa luồng suy luận CoT để đánh giá các ràng buộc không gian, chủ động phát hiện lỗi tràn số nguyên 64-bit và xây dựng khuôn mẫu I/O tùy chỉnh cho `__int128`.

2.3. Kiến trúc Đánh giá dựa trên Rubric

Để lượng hóa các phân tích định tính của CoT thành thước đo chuẩn xác, hệ thống áp dụng khung "Rubric Is All You Need" [22]. Khung này triết tiêu tính chủ quan của LLM thông qua cơ chế tính điểm trọng số nghiêm ngặt. Tổng điểm S (thang 100) được xác định qua Công thức (1):

$$S = w_1 \cdot C_{func} + w_2 \cdot C_{comp} + w_3 \cdot C_{opt} + w_4 \cdot C_{style} \quad (1)$$

- C_{func} (Tính năng): Tiêu chí này đánh giá tính đúng đắn của chương trình thông qua kết quả thực thi và khả năng xử lý các lỗi nghiêm trọng như `IndexError` hoặc phép chia cho 0. Nếu có dữ liệu từ hệ thống OJ, kết quả test-case cũng được sử dụng để kiểm chứng tính toàn vẹn logic của thuật toán. Đây là thành phần quan trọng nhất trong hệ thống đánh giá với trọng số $w_1 = 0,40$.

- C_{comp} (Hiệu suất thuật toán): Thành phần này tập trung đánh giá độ phức tạp tính toán của lời giải so với mức kỳ vọng của bài toán. Ví dụ, một chương trình sử dụng thuật toán $O(N^2)$ trong trường hợp yêu cầu tối ưu ở mức $O(N \log N)$ sẽ bị giảm đáng kể số điểm. Trọng số áp dụng cho tiêu chí này là $w_2 = 0,30$.

- C_{opt} (Tối ưu và xử lý ngoại lệ): Tiêu chí này xem xét khả năng kiểm soát các tình huống đặc biệt như tràn số nguyên, xử lý dữ liệu biên hoặc các nhánh điều kiện rỗng. Hệ thống cũng đánh giá mức độ sử dụng hợp lý các kiểu dữ liệu như `long long` hoặc `__int128` trong các bài toán xử lý số lớn. Trọng số tương ứng là $w_3 = 0,20$.

- C_{style} (Tính dễ đọc và bảo trì mã nguồn): Thành phần này được đánh giá thông qua độ phức tạp Cyclomatic (CC), quy tắc đặt tên biến và mức độ rõ ràng của tài liệu mã nguồn. Những

chương trình có giá trị $CC > 30$ sẽ bị xem là khó bảo trì và chịu điểm phạt tương ứng về mặt phong cách lập trình. Trọng số của tiêu chí này là $w_4 = 0,10$.

Đầu ra của mô hình LLM được chuẩn hóa dưới dạng JSON, bao gồm điểm thành phần, luồng suy luận CoT và các phản hồi giải thích. Cấu trúc này giúp hệ thống dễ dàng tích hợp với API và giao diện của các nền tảng OJ hiện có.

3. Thực nghiệm và đánh giá

Nhằm đánh giá hiệu quả của kiến trúc CoT trong bài toán chấm điểm và phân tích mã nguồn, nghiên cứu tiến hành thực nghiệm trên nhiều bài nộp lập trình và so sánh với các phương pháp LLM truyền thống. Phần này trình bày quy trình thực nghiệm, tiêu chí đánh giá và các kết quả thống kê đạt được, đồng thời phân tích một số tình huống tiêu biểu để làm rõ khả năng phát hiện lỗi, đánh giá độ phức tạp và hỗ trợ tối ưu hóa mã nguồn của hệ thống. Ngoài ra, nghiên cứu cũng xem xét bài toán đánh đổi giữa chất lượng phản hồi, độ trễ xử lý và chi phí triển khai khi áp dụng CoT trong thực tế.

3.1. Môi trường thực nghiệm và tập dữ liệu

Tập dữ liệu thực nghiệm đóng vai trò quan trọng trong việc đánh giá khả năng tổng quát hóa của hệ thống. Nghiên cứu sử dụng 1.250 bài nộp C/C++ được thu thập và ẩn danh từ các kỳ thi học sinh giỏi, môi trường lập trình thi đấu (tương đương Codeforces) và các học phần Cấu trúc Dữ liệu & Giải thuật. Dữ liệu được phân bổ thành ba nhóm chính: (1) Chuỗi Palindrome (vết cạn, Quy hoạch động, Manacher) nhằm kiểm tra lỗi IndexError; (2) Thao tác mảng và kỹ thuật hai con trỏ nhằm đánh giá khả năng phân tích độ phức tạp thuật toán (so sánh $\mathcal{O}(N)$ và $\mathcal{O}(N^2)$); và (3) Số nguyên tố và tổ hợp (Sàng Eratosthenes, lũy thừa ma trận, giới hạn $N \leq 10^{18}$) nhằm kiểm tra khả năng phát hiện tràn số và đề xuất `__int128`.

Để đảm bảo độ tin cậy của nhãn dữ liệu (ground truth), toàn bộ 1.250 bài nộp được gán nhãn thủ công bởi hai chuyên gia độc lập, với hệ số đồng thuận Cohen's Kappa đạt $\kappa = 0,86$. Nhằm đảm bảo tính minh bạch và hỗ trợ tái lập kết quả nghiên cứu (reproducibility), chúng tôi đã trích xuất, ẩn danh hóa và công bố một tập con tiêu biểu gồm 200 bài nộp tiêu biểu từ hệ thống [oj.hoccongnghie.com]. Tập dữ liệu này cùng với các tệp tin cấu trúc prompt chuẩn hóa, hiện đã được lưu trữ công khai tại kho dữ liệu: https://github.com/tuanlinhkinhte/TNU_ASG_Dataset. Việc công bố này không chỉ tạo điều kiện cho các nghiên cứu phái sinh mà còn khẳng định tính khách quan của tập dữ liệu thực nghiệm.

3.2. Phương pháp Đánh giá và Mô hình Cơ sở (Baseline)

Nghiên cứu xây dựng khung thực nghiệm A/B trên cùng một mô hình ngôn ngữ nền tảng (ví dụ GPT-4o) nhằm so sánh trực tiếp hiệu quả giữa hai phương pháp đánh giá mã nguồn tự động.

Zero-shot Prompting (Baseline): Trong kịch bản này, mô hình chỉ nhận thông tin cơ bản gồm mô tả bài toán và mã nguồn trước khi đưa ra đánh giá trực tiếp theo Rubric. Đây là cách tiếp cận phổ biến của nhiều hệ thống AGS tích hợp AI hiện nay.

CoT Prompting (Hệ thống đề xuất): Ở phương pháp đề xuất, mô hình được dẫn dắt thông qua kiến trúc prompt nhiều bước dựa trên kỹ thuật Chuỗi suy luận (CoT), bao gồm phân tích ngữ nghĩa chương trình, đánh giá độ phức tạp thuật toán, kiểm tra lỗi bộ nhớ và phát hiện nguy cơ tràn số nguyên.

Để đánh giá hiệu quả hệ thống, nghiên cứu sử dụng nhiều chỉ số định lượng quan trọng. *Pass@1 Accuracy* đo lường khả năng dự đoán đúng kết quả của mô hình so với phán quyết từ hệ thống sandbox. Hidden Bug Recall được dùng để đánh giá khả năng phát hiện các lỗi tiềm ẩn như IndexError và lỗi tràn số nguyên. Trong khi đó, Suggestion Precision phản ánh độ chính xác của các gợi ý tối ưu hóa, đặc biệt là việc đề xuất sử dụng `__int128` khi cần thiết. Nghiên cứu cũng phân tích lượng token tiêu thụ và độ trễ xử lý nhằm xem xét khả năng triển khai thực tế của hệ thống.

Bên cạnh các chỉ số hiệu năng, kiểm định thống kê McNemar được áp dụng để đánh giá ý nghĩa của sự khác biệt giữa hai phương pháp trên các tác vụ phân loại nhị phân như *Pass@1* và khả năng phát hiện *IndexError*. Giá trị $p < 0,05$ được sử dụng làm ngưỡng xác định ý nghĩa thống kê cho các kết quả thực nghiệm.

Nhằm đảm bảo tính minh bạch và khả năng tái lập (reproducibility) của quy trình thực nghiệm, nghiên cứu thiết lập cấu hình đồng nhất cho mô hình ngôn ngữ lớn trên toàn bộ tập dữ liệu mẫu. Cụ thể, hệ thống sử dụng mô hình nền tảng GPT-4o (phiên bản thương mại mã hiệu gpt-4o-2024-05-13) thông qua giao diện lập trình ứng dụng (OpenAI API). Để triệt tiêu tính ngẫu nhiên, tăng cường độ nhất quán của phản hồi và hạn chế tối đa hiện tượng ảo giác (hallucination) trong quá trình chấm điểm tự động, tham số nhiệt độ (temperature) được đặt cố định ở mức 0.0. Đồng thời, nhằm chuẩn hóa cấu trúc đầu ra theo hệ thống tiêu chí định lượng, cơ chế phản hồi định dạng dữ liệu cấu trúc (JSON Mode) được kích hoạt bắt buộc. Chi tiết về các tham số vận hành của mô hình LLM được tổng hợp cụ thể trong Bảng 1.

Bảng 1. Cấu hình tham số kỹ thuật và môi trường vận hành LLM

Tham số kỹ thuật	Giá trị thiết lập	Mục đích sử dụng/ Kỹ thuật
Mô hình nền tảng (Model)	gpt-4o-2024-05-13	Đảm bảo hiệu năng lập luận đa tầng ổn định
Giao thức kết nối	OpenAI API	Tích hợp hệ thống tự động
Nhiệt độ (Temperature)	0.0	Triệt tiêu tính ngẫu nhiên, đảm bảo tính nhất quán
Top-p Sampling	1.0	Giữ nguyên phân phối xác suất từ phân tích kỹ thuật
Giới hạn đầu ra (Max Tokens)	2.048 token	Đảm bảo không gian suy luận đầy đủ cho chuỗi CoT
Chế độ định dạng (Format)	JSON Mode	Chuẩn hóa cấu trúc Rubric phục vụ trích xuất tự động
Cơ chế kiểm soát	Thẻ <thinking>	Ép luồng suy luận tuân thủ trước khi chấm điểm

Việc cấu hình chặt chẽ các tham số này đóng vai trò như một bộ lọc kỹ thuật, giúp lượng hóa chính xác các phân tích định tính từ luồng suy luận CoT thành điểm số Rubric mà không bị ảnh hưởng bởi sự bất định của mô hình tạo sinh.

3.3. Kết quả định lượng

Các kết quả thực nghiệm được tổng hợp trong Bảng 2 cho thấy kiến trúc CoT đạt hiệu quả vượt trội so với phương pháp Baseline trên hầu hết các tiêu chí đánh giá. Sự cải thiện không chỉ thể hiện ở độ chính xác trong chấm điểm mà còn ở khả năng phân tích thuật toán, phát hiện lỗi tiềm ẩn và đưa ra các gợi ý tối ưu hóa phù hợp với bối cảnh lập trình thi đấu.

Bảng 2. So sánh hiệu năng phát hiện lỗi, đánh giá thuật toán và độ chính xác dự đoán giữa phương pháp Zero-shot và Chuỗi suy luận (CoT) trên 1.250 bài nộp

Tiêu chí đánh giá	Phương pháp Zero-shot (Cơ sở)	Phương pháp CoT (Đề xuất)	Mức cải thiện
Độ chính xác Pass@1	76,4%	85,1%	+8,7%
Nhận diện độ phức tạp (O)	82,3%	91,5%	+9,2%
Độ thu hồi lỗi IndexError (Mảng/Chuỗi)	62,1%	84,3%	+22,2%
Độ thu hồi lỗi tràn số nguyên	58,5%	79,8%	+21,3%
Độ chính xác đề xuất int128	31,0%	88,6%	+57,6%

Độ chính xác tổng thể và phân tích thuật toán: Kết quả thực nghiệm cho thấy kiến trúc CoT cải thiện 8,7% độ chính xác *Pass@1* so với phương pháp zero-shot (kiểm định McNemar, $p < 0,001$), đồng thời tăng 9,2% khả năng nhận diện đúng độ phức tạp thuật toán ($p < 0,01$). Mức cải thiện này cho thấy hiệu quả rõ rệt của quá trình suy luận nhiều bước. Trong khi mô hình zero-shot thường gặp khó khăn với các lời gọi hàm có chi phí tính toán ẩn như *sort()*, cơ chế phân tích luồng điều khiển của CoT giúp mô hình suy luận chính xác hơn về giới hạn thời gian thực thi.

Phát hiện lỗi biên (IndexError): Một trong những cải tiến đáng chú ý nhất của CoT nằm ở khả năng phát hiện lỗi truy cập vượt biên. Với các chương trình có cấu trúc vòng lặp phức tạp,

mô hình zero-shot thường không đối chiếu chính xác biến lập với kích thước mảng, khiến Recall chỉ đạt 62,1%. Nhờ kỹ thuật “dry-run” mô phỏng quá trình thực thi, CoT có thể phát hiện hiệu quả các lỗi off-by-one và nâng Recall lên 84,3%, tương ứng mức cải thiện 22,2% ($p < 0,001$).

Tối ưu tràn số và nhận diện nhu cầu sử dụng __int128: Sự khác biệt giữa hai phương pháp thể hiện rõ trong các bài toán tổ hợp và xử lý số lớn. Phương pháp zero-shot thường đề xuất __int128 chưa phù hợp ở các bài toán nhỏ ($N \leq 10^5$), đồng thời bỏ sót các trường hợp có nguy cơ tràn số trong phép nhân trung gian, khiến độ chính xác chỉ đạt 31,0%. Ngược lại, kiến trúc CoT yêu cầu mô hình đối chiếu phạm vi giá trị của biến với ngưỡng 9×10^{18} , từ đó nâng độ chính xác đề xuất lên 88,6% ($p < 0,001$). Kết quả này cho thấy hệ thống có khả năng đưa ra khuyến nghị phù hợp hơn đối với các bài toán xử lý số nguyên lớn trong lập trình thi đấu.

3.4. So sánh SOTA, Phân tích Cắt bỏ (Ablation Study) và Đánh giá Baseline mở rộng

Để làm rõ những đóng góp đột phá của hệ thống so với các hướng nghiên cứu hiện hành (State-of-the-Art - SOTA), nghiên cứu tiến hành đối chiếu định tính với hai framework AGS tích hợp LLM tiên tiến đã được đề cập là CodEv và StepGrade. Bảng 3 tổng hợp sự khác biệt về mặt kiến trúc và khả năng xử lý các tình huống đặc thù trong lập trình thi đấu.

Bảng 3. So sánh định tính các tính năng cốt lõi giữa hệ thống CoT đề xuất và các phương pháp SOTA

Tiêu chí đối chiếu	CodEv [10]	StepGrade [11]	Hệ thống CoT đa tầng (Đề xuất)
Cơ chế suy luận cốt lõi	Bỏ phiếu (Voting) đa tác nhân	Phân tích nhận thức ngữ cảnh (Context-aware)	Chuỗi suy luận tuần tự (Structured CoT)
Trọng tâm đánh giá	Giảm thiểu hiện tượng ảo giác (hallucination)	Đánh giá cấu trúc dữ liệu và độ phức tạp thuật toán	Phân tích thuật toán, ranh giới bộ nhớ và tối ưu tài nguyên
Phát hiện lỗi truy cập biên (IndexError)	Bỏ sót phần lớn, phụ thuộc vào kết quả trả về từ Hộp cát (Sandbox)	Nhận diện dựa trên phân tích mã tĩnh cơ bản	Tích hợp kỹ thuật mô phỏng "dry-run" vòng lặp để dự báo chủ động
Phòng ngừa tràn số nguyên (Overflow)	Không có cơ chế can thiệp chuyên biệt	Đánh giá giới hạn cấu trúc dữ liệu chung	Mô-đun quy tắc ngầm: Ước lượng độ lớn biến và tự động đề xuất ép kiểu <i>int128</i>

Như được thể hiện trong Bảng 3, trong khi CodEv tập trung vào cơ chế bỏ phiếu để giảm hiện tượng ảo giác và StepGrade tối ưu hóa việc phân tích cấu trúc dữ liệu, cả hai hệ thống này vẫn thiếu các cơ chế chuyên biệt để xử lý những ràng buộc tài nguyên nghiêm ngặt. Hệ thống CoT đề xuất khóa lấp khoảng trống này thông qua hai đột phá kỹ thuật chính: thứ nhất, ứng dụng kỹ thuật mô phỏng "dry-run" ở Bước 3 giúp phát hiện sớm rủi ro IndexError mà phân tích tĩnh thông thường dễ bỏ sót; thứ hai, mô-đun quy tắc ngầm ở Bước 4 có khả năng can thiệp chủ động, ép kiểu __int128 đối với các bài toán có tích trung gian vượt ngưỡng 9×10^{18} . Đây là những cam bẫy thực thi rất đặc thù trong C/C++ mà các hệ thống SOTA hiện tại chưa hỗ trợ toàn diện.

Baseline mở rộng (Few-shot Prompting): Trong thiết lập này, mô hình được cung cấp ba ví dụ tham chiếu nhưng không có hướng dẫn suy luận theo từng bước. Kết quả cho thấy độ chính xác Pass@1 tăng nhẹ từ 76,4% lên 79,2% so với zero-shot, song vẫn thấp hơn đáng kể so với mức 85,1% của kiến trúc CoT. Điều này cho thấy hiệu quả của hệ thống chủ yếu đến từ cơ chế suy luận có cấu trúc thay vì khả năng học theo mẫu đơn thuần.

Phân tích cắt bỏ các mô-đun CoT: Để đánh giá vai trò của từng bước trong chuỗi suy luận, nghiên cứu triển khai hai biến thể rút gọn của hệ thống:

- **Biến thể A (Loại bỏ Bước 1 – Phân tích ngữ nghĩa):** Khi thiếu giai đoạn phân tích luồng điều khiển và cấu trúc chương trình, khả năng nhận diện độ phức tạp thuật toán giảm từ 91,5% xuống còn 83,1%. Kết quả này cho thấy việc hiểu ngữ nghĩa mã nguồn có vai trò quan trọng trong quá trình suy luận về hiệu năng thuật toán.

○ **Biến thể B (Loại bỏ Bước 4 – Kiểm tra tràn số):** Khi cơ chế phân tích giới hạn số nguyên bị loại bỏ, độ chính xác trong việc đề xuất sử dụng `__int128` giảm mạnh từ 88,6% xuống 42,4%. Mô hình có xu hướng đề xuất kiểu dữ liệu 128-bit không cần thiết ở các bài toán đơn giản, đồng thời bỏ sót các trường hợp có nguy cơ tràn số trong phép tính trung gian.

Kết quả ở Bảng 4 cho thấy hiệu năng của hệ thống suy giảm rõ rệt khi thiếu các bước suy luận quan trọng, qua đó khẳng định tính hỗ trợ giữa các mô-đun trong kiến trúc CoT đề xuất.

Bảng 4. Phân tích cắt bỏ (Ablation study) và hiệu năng của mô hình cơ sở mở rộng, minh họa sự suy giảm hiệu năng khi loại bỏ các bước suy luận CoT cụ thể.

Biến thể kiến trúc	Độ chính xác Pass@1	Nhận diện độ phức tạp (O)	Độ chính xác đề xuất <code>__int128</code>
Phương pháp Zero-shot (Cơ sở)	76,4%	82,3%	31,0%
Phương pháp Few-shot (3-shot)	79,2%	84,5%	38,2%
CoT không có Bước 1 (Không giải mã ngữ nghĩa)	80,5%	83,1%	86,4%
CoT không có Bước 4 (Không có quy tắc ngầm)	84,2%	90,8%	42,4%
Phương pháp CoT đầy đủ (Đề xuất)	85,1%	91,5%	88,6%

3.5. Phân tích tình huống định tính

Bên cạnh các kết quả định lượng, giá trị sự phạm cốt lõi của hệ thống được thể hiện qua năng lực chẩn đoán và giải thích lỗi. Bảng 5 minh họa trực quan cách kiến trúc CoT vượt trội hơn hẳn phương pháp Zero-shot trong việc xử lý các cạm bẫy đặc thù của lập trình thi đấu, đồng thời nhìn nhận khách quan giới hạn của công nghệ phân tích tĩnh dựa trên LLM.

Bảng 5. So sánh kết quả phân tích và phản hồi giữa phương pháp Zero-shot và hệ thống CoT

Loại lỗi (Tình huống đặc thù)	Phản hồi của phương pháp Zero-shot (Baseline)	Phản hồi và Can thiệp của hệ thống CoT (Đề xuất)
Tình huống 1: Tràn số nguyên trung gian Bài toán tổ hợp sử dụng biến long long ($N \approx 10^{18}$): $res = (res * prime_factor) \% MOD$;	Đánh giá lời giải hợp lệ dựa trên độ phức tạp thời gian $O(\sqrt{N})$. Bỏ sót hoàn toàn nguy cơ tràn số nguyên trong phép nhân trung gian.	Phân tích: Nhận diện tích res * prime_factor sẽ vượt ngưỡng giới hạn 9×10^{18} của kiểu 64-bit trước khi thực hiện phép chia lấy dư. Can thiệp: Giải thích bản chất lỗi và đề xuất ép kiểu trung gian: $res = (long_long)((_int128)res * prime_factor \% MOD)$;
Tình huống 2: Truy cập vượt biên (IndexError) Thuật toán Hai con trỏ có chứa điều kiện nhìn trước (look-ahead): $if (str[left+1] == str[right-1])$	Bỏ qua rủi ro truy cập bộ nhớ chưa cấp phát khi biến lặp tiến sát ranh giới cuối mảng.	Phân tích: Cơ chế "dry-run" phát hiện <code>str[left+1]</code> sẽ gây lỗi Undefined Behavior hoặc Segmentation Fault khi left chạm tới cuối chuỗi. Can thiệp: Yêu cầu bổ sung chốt chặn an toàn trước khi truy cập: $if (left + 1 < str.length())$
Tình huống 3: Tràn ngăn xếp ngấm (Điểm mù của hệ thống) Thuật toán đệ quy sâu (DFS) trên đồ thị cây suy biến với $N = 10^5$ đỉnh.	Bỏ sót lỗi, đánh giá chương trình thực thi thành công.	Phân tích (Thất bại chẩn đoán): Tính toán chuẩn xác độ phức tạp tường minh là $O(N)$ và dự đoán mã đúng, nhưng không nhận thức được giới hạn ngấm của bộ nhớ call-stack gây lỗi Stack Overflow trong thực tế. Kết luận: Khẳng định giới hạn của LLM thuần túy và sự bắt buộc phải duy trì Hộp cát (Sandbox) thực thi động.

Từ thất bại trong chẩn đoán ở Tình huống 3, có thể thấy việc đánh giá các lỗi liên quan đến giới hạn phân cứng ngàm định (như dung lượng call-stack) nằm ngoài khả năng suy luận thuần túy của LLM. Để giải quyết triệt để điểm mù này, hệ thống cần được tích hợp thêm module phân tích Cây cú pháp trừu tượng (Abstract Syntax Tree - AST) nhằm đánh giá tính độ sâu đệ quy, hoặc thiết lập cơ chế đồng bộ hóa trực tiếp để bắt tín hiệu ngắt phân trang bộ nhớ (SIGSEGV / SIGFPE) từ môi trường Hộp cát (Sandbox) trả về, qua đó cung cấp bối cảnh xác đáng hơn cho AI sinh phản hồi.

3.6. Bài toán Đánh đổi - Chi phí Tài chính và Khả năng Triển khai Quy mô lớn

Việc tích hợp cơ chế suy luận nhiều bước dựa trên LLM mang lại những cải thiện đáng kể về khả năng phân tích và phản hồi sự phạm, tuy nhiên cũng kéo theo sự gia tăng về chi phí tính toán và thời gian xử lý. Bảng 6 trình bày các chỉ số trung bình liên quan đến lượng tài nguyên sử dụng và độ trễ.

Bảng 6. So sánh chi phí token và độ trễ giữa các kỹ thuật chấm điểm trên hạ tầng API của LLM (ví dụ: GPT-4o)

Tiêu chí đánh giá	Phương pháp Zero-shot	Phương pháp CoT	Mức chênh lệch
Số token đầu ra trung bình	~450 token	~1.150 token	2,55x
Độ trễ trung bình	2,4 giây	5,8 giây	+3,4 giây

Dựa trên mức tiêu thụ token ở Bảng 6 và đơn giá API GPT-4o tiêu chuẩn (5 USD/1 triệu token đầu vào, 15 USD/1 triệu token đầu ra), chi phí vận hành cho 1.000 bài nộp bằng phương pháp Zero-shot ước tính khoảng 14,25 USD. Trong khi đó, kiến trúc CoT đầy đủ tiêu tốn khoảng 24,75 USD (tăng 73,7%). Cùng với độ trễ phản hồi lên đến 5,8 giây, việc áp dụng CoT thuần túy sẽ gặp rủi ro nghẽn cổ chai (bottleneck) do giới hạn tần suất gọi API (Rate Limits), đặc biệt khi hệ thống phải xử lý lượng truy cập đồng thời lớn như trong các kỳ thi học kỳ tập trung tại trường đại học hay các đợt tuyển sinh tại trung tâm đào tạo lập trình.

Nhằm giải quyết bài toán quy mô, nghiên cứu đề xuất triển khai hệ thống theo Kiến trúc Lai (Hybrid Architecture) có điều kiện. Cụ thể, hệ thống Hộp cát (Sandbox) truyền thống được sử dụng làm bộ lọc tuyến đầu, thực thi mã trong thời gian $\approx 0,1$ giây để phán quyết Đúng/Sai ban đầu. Luồng suy luận CoT chuyên sâu chỉ được kích hoạt khi: (1) Sandbox trả về lỗi logic (WA, TLE, RTE), hoặc (2) người học chủ động yêu cầu giải thích sự phạm. Dựa trên tỷ lệ bài nộp lỗi hoặc cần hỗ trợ trung bình khoảng 25% trong thực tế giảng dạy, chi phí vận hành cho 1.000 bài nộp giảm mạnh xuống chỉ còn 6,19 USD. Cách tiếp cận phân luồng bất đồng bộ này không chỉ tối ưu 75% chi phí so với CoT đại trà, mà còn giải phóng băng thông máy chủ, khẳng định tính khả thi tuyệt đối khi triển khai AGS thông minh ở quy mô lớn.

4. Kết luận

Nghiên cứu này đã đề xuất và hiện thực hóa thành công giải pháp tích hợp mô hình ngôn ngữ lớn vào hệ thống chấm điểm tự động (AGS) thông qua kiến trúc Chuỗi suy luận (CoT) đa tầng. Tiếp cận này định hình lại phương thức đánh giá lập trình truyền thống, dịch chuyển từ phán quyết đúng/sai nhị phân sang cung cấp hệ thống phản hồi sự phạm chuyên sâu về ngữ nghĩa, độ phức tạp thuật toán và ranh giới tài nguyên. Kết quả thực nghiệm trên 1.250 bài nộp chứng minh tính hiệu quả vượt trội của giải pháp đề xuất so với baseline zero-shot. Sự cải thiện được lượng hóa rõ rệt qua các chỉ số: tăng 8,7% độ chính xác Pass@1, nâng 9,2% năng lực nhận diện độ phức tạp Big-O, đồng thời tối ưu hóa khả năng phát hiện lỗi IndexError thêm 22,2% và đạt mức chính xác vượt trội 57,6% khi đưa ra khuyến nghị sử dụng kiểu dữ liệu mở rộng `__int128`.

Bên cạnh các đóng góp thực tiễn, nghiên cứu thẳng thắn chỉ ra hai giới hạn cốt lõi: điểm mù chẩn đoán đối với lỗi tràn bộ nhớ ngàm định (Stack Overflow) do LLM thiếu nhận thức về giới hạn vật lý của phần cứng, và mức độ phụ thuộc cao vào kỹ thuật thiết kế prompt thủ công. Để giải quyết các thách thức này, lộ trình nghiên cứu tương lai sẽ tập trung theo hai hướng chiến

lược. Về mặt kiến trúc, hệ thống sẽ kết hợp luồng suy luận CoT với phân tích tĩnh Cây cú pháp trừu tượng (AST) để kiểm soát độ sâu đệ quy, đồng thời đồng bộ hóa trực tiếp tín hiệu ngắt SIGSEGV từ Hộp cát (Sandbox) thực thi động. Về mặt tối ưu hóa mô hình, việc tích hợp các framework tự động hóa prompt (như DSPy) cùng với việc tinh chỉnh các mô hình ngôn ngữ nhỏ (SLM) chuyên biệt sẽ được triển khai, hướng tới xây dựng một hệ thống gia sư thông minh toàn diện, tối ưu chi phí vận hành và có năng lực tổng quát hóa cao trên đa ngôn ngữ lập trình.

TÀI LIỆU THAM KHẢO/ REFERENCES

- [1] Y. Jung, Y. Lee, J. Bae, D. Y. Kim, H. Choi, M. Kang, and U. Lee, "Exploring the Role of Automated Feedback in Programming Education: A Systematic Literature Review," *arXiv preprint arXiv:2602.00089*, 2026, doi: 10.48550/arXiv.2602.00089.
- [2] M. Messer, N. C. C. Brown, M. Kölling, and M. Shi, "Automated Grading and Feedback Tools for Programming Education: A Systematic Review," *ACM Transactions on Computing Education*, vol. 24, no. 1, 2024, doi: 10.1145/3636515.
- [3] M. Al-Shakarji, "Mitigating Hallucination in Large Language Model Code Generation for Higher Education," Degree Project in Computer Science and Engineering, KTH Royal Institute of Technology, 2024.
- [4] M. Richards, K. Waugh, M. Slaymaker, M. Petre, J. Woodthorpe, and D. Gooch, "Bob or bot: Exploring ChatGPT's answers to university computer science assessment," *ACM Transactions on Computing Education*, vol. 24, no. 1, pp. 1-32, 2024, doi: 10.1145/3633287.
- [5] B. Althani, "A Multi-Objective Statistical Framework for Evaluating LLM-Based Code Modernization: Transformation Pattern Analysis and Effect Size Validation," *Computers*, vol. 15, no. 3, 2026, Art. no. 148, doi: 10.3390/computers15030148.
- [6] M. Mahdaoui, S. Nouh, M. S. E. K. Alaoui, and K. Kandali, "Automated Grading Method of Python Code Submissions Using Large Language Models and Machine Learning," *Information*, vol. 16, no. 8, 2025, Art. no. 674, doi: 10.3390/info16080674.
- [7] A. Memon and A. Mohamed, "Evaluating Generative AI for CS1 Code Grading: Direct Vs Reverse Methods," in *Proceedings of the IEEE International Conference on Collaborative Advances in Software and Computing (CASCON)*, 2025, pp. 178-183.
- [8] B. Cowan, Y. Watanobe, and A. Shirafuji, "Enhancing Programming Learning with LLMs: Prompt Engineering and Flipped Interaction," in *Proceedings of the 2023 4th Asia Service Sciences and Software Engineering Conference*, 2023, pp. 10-16.
- [9] L. Y. Tan, S. Hu, D. J. Yeo, and K. H. Cheong, "A Comprehensive Review on Automated Grading Systems in STEM Using AI Techniques," *Mathematics*, vol. 13, no. 17, 2025, Art. no. 2828, doi: 10.3390/math13172828.
- [10] E.-Q. Tseng, P.-C. Huang, C. Hsu, P.-Y. Wu, C.-T. Ku, and Y. Kang, "CodEv: An Automated Grading Framework Leveraging Large Language Models for Consistent and Constructive Feedback," *arXiv preprint arXiv:2501.10421*, 2025, doi: 10.48550/arXiv.2501.10421.
- [11] M. Akyash, K. Z. Azar, and H. M. Kamali, "StepGrade: Grading Programming Assignments with Context-Aware LLMs," in *Proc. 2025 IEEE Integrated STEM Education Conference (ISEC)*, 2025, pp. 1-6, doi: 10.1109/ISEC64801.2025.11147374.
- [12] A. Tabassum, M. S. Hossain, M. F. Arefin, T. Islam, and T. S. Zaman, "A-ProS: Towards Reliable Autonomous Programming Through Multi-Model Feedback," *arXiv preprint arXiv:2605.18073*, 2026, doi: 10.48550/arXiv.2605.18073.
- [13] S. Elnaffar, F. Rashidi, and A. Z. Abualkishik, "Teaching with AI: A Systematic Review of Chatbots, Generative Tools, and Tutoring Systems in Programming Education," *arXiv preprint arXiv:2510.03884*, 2025, doi: 10.48550/arXiv.2510.03884.
- [14] K. Wang, Y. Shi, R. Shen, T. W. Price, and D. Wang, "CodeExemplar: Example-Based Scaffolding for Introductory Programming in the GenAI Era," *arXiv preprint arXiv:2603.23830*, 2026, doi: 10.48550/arXiv.2603.23830.
- [15] IBM, "What is chain of thought (CoT) prompting?," IBM Think, 2024. [Online]. Available: <https://www.ibm.com/think/topics/chain-of-thoughts?lnk=thinkhpeverpe4us>. [Accessed May 23, 2026].

-
- [16] S. Halim, F. Halim, and S. Prasetya, *Competitive Programming 4: The Lower Bound of Programming Contests in the 2020s*, 4th ed. LuLu Press, 2020.
- [17] N. Hossain and M. Martonosi, "TEMpesT: Testing Empirically for Memory Transistency," *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 4, 2025, Art. no. 165, doi: 10.1145/3774419.
- [18] J. Schultke, "Bit-precise integers," ISO/IEC JTC1/SC22/WG21 C++ Standard Committee Paper, no. P3666R4, 2026. [Online]. Available: <https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2026/p3666r4.html>. [Accessed May 23, 2026].
- [19] Free Software Foundation, "Using the GNU Compiler Collection (GCC): 128-bit Integers," GNU Project, 2024. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/005f_005fint128.html. [Accessed May 23, 2026].
- [20] ISO/IEC, "Information technology — Programming languages — C++," ISO/IEC 14882:2020 Standard, 2020. [Online]. Available: <https://www.iso.org/standard/79358.html>. [Accessed May 23, 2026].
- [21] J. Cambronoero, S. Mukherjee, T. Mytkowicz, and G. Loop, "Fault-Aware Neural Code Rankers," Microsoft Research, 2022. [Online]. Available: <https://www.microsoft.com/en-us/research/publication/fault-aware-neural-code-rankers/>. [Accessed May 23, 2026].
- [22] J. Yoon, S. Lee, H. Chang, and S. Kim, "Rubric Is All You Need: Improving LLM-based Code Evaluation With Question-Specific Rubrics," *arXiv preprint arXiv:2503.23989*, 2025, doi: 10.48550/arXiv.2503.23989.