JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 27, No. 10, pp. 2991-2998, Oct. 2026

Check for updates

비전공자를 위한 LLM 기반 코드 생성 활용 및 평가 프레임워크 설계 및 검증

정 인 철[1] · 김 태 홍[2*] · 유 재 수[2]
[1]충북대학교 전자정보공학과 석사과정
[2]충북대학교 정보통신공학부 및 컴퓨터정보통신연구소 교수

Design and Validation of an LLM-Based Framework for Code Generation and Evaluation for Non-Computer Science Users

Incheol Jeong[1] · Taehong Kim[2*] · Jaesoo Yoo[2]

[1]Master's Course, Dept. of Electronics and Computer Eng., Chungbuk National University, Cheongju 28644, Korea
[2]Professor, School of Information and Communication Eng., Chungbuk National University, Cheongju 28644, Korea

[요 약]

대형 언어 모델(LLM: large language model)의 발전은 비전공자의 소프트웨어 개발 접근성을 높여 '시티즌 디벨로퍼(Citizen Developer)' 흐름을 만들었으나, 생성된 코드의 신뢰성을 검증하고 오류를 교정하는 체계적 가이드라인은 부재하다. 본 연구는 문제 선정, 프롬프트 입력, 코드 생성, 실행·평가, 피드백 수집, 오류 교정의 6단계 순환 프레임워크를 제안·검증한다. LeetCode 최신 문제 180개로 구성된 성능평가에서는 ChatGPT 5.3, Google Gemini 3 Flash, GitHub Copilot을 비교하였으며, Gemini 3 Flash가 정답률 93.3%로 가장 높은 안정성을 보였다. 이어진 제거 연구(Ablation Study)에서 1회 시도 정답률(Pass@1)이 40%에서 80%로 향상되고 구문·실행 오류가 통제되었으며, 평균 실행 효율과 메모리 효율이 각각 11.8%p, 19.5%p 개선되었음을 검증하였다. 이는 구조화된 평가 절차를 통해 비전공자의 AI 코드 생성 활용 효율을 개선할 수 있음을 보여준다.

[Abstract]

The advancement of large language models (LLMs) has lowered the barrier to software development for non-computer science users, giving rise to the concept of the "citizen developer." However, systematic guidelines for verifying AI-generated code and correcting errors remain lacking. This study proposes and validates a six-step iterative framework consisting of problem selection, prompt input, code generation, code execution and evaluation, feedback collection, and error correction. Using 180 recent LeetCode problems, the performance of ChatGPT 5.3, Google Gemini 3 Flash, and GitHub Copilot was compared, with Gemini 3 Flash achieving the highest accuracy at 93.3%. A subsequent ablation study demonstrated that the proposed framework doubled the zero-shot pass@1 rate from 40% to 80%, reduced syntax and runtime errors, and improved runtime and memory efficiency by 11.8 percentage points and 19.5 percentage points, respectively. These findings demonstrate that a structured evaluation procedure can help non-computer science users utilize AI-generated code more efficiently.

색인어 : 대형 언어 모델, 코드 생성, 시티즌 디벨로퍼, 평가 프레임워크, 프롬프트 엔지니어링
Keyword : Large Language Model, Code Generation, Citizen Developer, Evaluation Framework, Prompt Engineering

http://dx.doi.org/10.9728/dcs.2026.27.10.2991

This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-CommercialLicense(http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Received 15 July 2026; **Revised** 10 August 2026
Accepted 24 August 2026

***Corresponding Author; Taehong Kim**

Tel: +82-43-261-2481
E-mail: taehongkim@chungbuk.ac.kr

Ⅰ. 서 론

인공지능 기술의 비약적 발전, 특히 대형 언어 모델(LLM: large language model)의 등장은 소프트웨어 개발 방식의 근본적인 패러다임 전환을 야기하고 있다. 과거의 프로그래밍은 인간이 기계의 언어를 학습해 일방적으로 명령을 내리는 수직적 구조였다면, 현재는 기계가 인간의 자연어를 맥락적으로 이해하고 실행 가능한 코드로 변환하는 상호작용적 개발 패러다임으로 진화하고 있다. 이러한 변화의 중심에는 방대한 소스 코드를 학습해 논리적 추론과 코드 생성 능력을 갖춘 LLM이 자리하고 있으며, 이는 기술적 전문성의 진입 장벽을 획기적으로 낮추는 계기가 되었다.

최근 이러한 기술의 발전은 교육 현장과 산업 전반에 전문적인 개발 교육을 받지 않고도 소프트웨어를 구축하는 '시티즌 디벨로퍼(Citizen Developer)'라는 새로운 흐름을 만들어 내고 있다. 과거의 컴퓨터 과학 교육이 복잡한 문법을 익히고 암기하는 데 초점을 맞췄다면, 이제는 다양한 도구를 써서 문제를 정의하고 해결 방안을 설계하는 분석적 사고력과 문제 해결 능력이 중요한 역량으로 떠오르고 있다[1]. 특히 프로그래밍이 익숙하지 않은 초보자들에게 LLM 기반 코드 생성 도구는 단순히 문법을 도와주는 수준을 넘어서, 가독성과 구조적 일관성을 갖춘 코드를 생성할 수 있도록 지원하는 유용한 도구로 자리 잡고 있다[2].

하지만 기술 접근 장벽이 낮아졌음에도 불구하고, 실제 활용 현장에서는 비전공자들이 생성된 코드의 '기술적 신뢰성'과 이를 업무에 적용하는 '운용 프로세스'에 있어 큰 혼란을 겪는 경우가 많다. 현재 시장에는 ChatGPT, Gemini 등 다양한 특성을 지닌 LLM 도구가 존재하지만, 비전공자가 자신의 목적에 가장 적합한 도구를 선택하고, 생성된 결과물을 객관적으로 평가할 수 있는 기준은 거의 마련되어 있지 않다.

무엇보다 현재 학계에서 발표된 LLM 관련 연구들은 대부분 단일 벤치마크 점수처럼 기능적 정확성 평가에 집중되어 있다[3]. 게다가 이런 연구들은 주로 전문 개발자의 실무 환경만을 전제로 하기 때문에, 프로그래밍 경험이 적은 초보자나 비전공자가 인공지능(AI: artificial intelligence)을 활용하고 코드를 평가하는 방법에 대한 논의는 여전히 부족하다[1],[2]. 실제로 최근 사용자 연구에서도 AI 코딩 도구를 활용할 때, 프로그래밍 전문 지식이 부족한 사용자는 생성된 코드의 로직을 파악하고 디버깅하는 데 상당한 어려움을 겪는다는 점이 실증적으로 확인된 바 있다[4].

이에 본 연구는 비전공자가 LLM 기반 코드 생성 도구를 활용하여 알고리즘 문제를 해결하는 전 과정을 체계적으로 정리하고, 결과를 다양한 관점에서 평가할 수 있는 통합 프레임워크를 설계하고 검증하는 것을 목표로 한다. 이를 위해 비전공자 중심의 실무 활용 프레임워크를 제안하고, 대표적인 생성형 AI 3종의 난이도별 성능을 비교 분석하며, 정량적·정성적 지표를 통해 프레임워크의 실효성을 검증한다.

Ⅱ. 관련 연구

2-1 AI 코드 생성 도구 개요

본 연구에서는 대표적인 AI 코드 생성 도구로 ChatGPT, Google Gemini, GitHub Copilot을 비교 대상으로 선정한다. 세 모델은 각각 범용 대화형 모델, 멀티모달 기반 모델, 그리고 개발 환경에 특화된 모델이라는 각기 다른 특징을 가지고 있으며, 실제 개발 현장에서 널리 활용되고 있다. 최근 학계에서도 대형 언어 모델을 소프트웨어 공학 전반에 적용하기 위한 다양한 리뷰 연구가 활발히 진행되고 있으며[5], 특히 프로그래밍 초보자가 자연어 프롬프트를 활용해 코딩 문제를 해결하는 방식[6]이나, 프롬프트의 구조가 AI 코드 생성 결과물에 미치는 영향[7] 등 프롬프트 엔지니어링의 실효성을 검증하는 연구들이 지속적으로 보고되고 있다. 본 연구에서 선정한 AI 코드 생성 도구 3종의 주요 특징은 표 1과 같다.

표 1. AI 코드 생성 도구 3종 비교
Table 1. Comparison of three AI code generation tools

Category	ChatGPT 5.3	Gemini 3 Flash	GitHub Copilot
Provider	OpenAI	Google	Microsoft & GitHub
Interface	General-purpose Chatbot	Multimodal AI	Code Assistant (IDE)
Environment	Web Browser	Web Browser	VS Code / IDE Built-in
Key Strength	Reasoning & Debugging	Fast Multimodal Processing	Real-Time Code Completion

2-2 LLM 기반 코드 생성 연구 동향 및 차별점

최근 소프트웨어 공학 분야에서는 LLM을 활용한 코드 생성 능력을 평가하기 위한 연구가 활발히 진행되고 있다. 대표적으로 HumanEval 벤치마크를 활용하여 다양한 AI 모델의 Python 코드 생성 정확도를 비교 분석한 연구[3]가 다수 발표되었으며, 최근에는 단순히 코드가 작동하는지를 넘어 실행 시간과 메모리 사용량 등 '코드의 자원 효율성'까지 엄격하게 평가하려는 새로운 연구 동향[8],[9]도 등장하고 있다.

그러나 지금까지의 선행 연구에는 공통된 한계가 있다. 대부분의 연구가 'AI 모델의 벤치마크 성능'이나 기계적으로 정확한 코드 작성에 치중하거나, 코드를 직접 검증하고 수정할 수 있는 '전문 개발자'의 시각만을 중점적으로 다뤘다는 점이다. 프로그래밍 지식이 부족한 비전공자 입장에서는 코드를 직접 작성하는 것 자체가 가장 먼저 부딪히는 큰 장벽이며, AI 도구 덕분에 코드 생성 자체의 진입 장벽은 크게 낮아졌지만, 처음부터 어떻게 요청해야 하는지, 겉으로는 정상적으로 보이는 코드가 원하는 대로 동작하는지 평가하는 방법, 오류가 났을 때 어떻게 고쳐야 할지 등의 검증 및 교정 과정은 비전공자에게는 코드 작성만큼이나 어렵고 부담이 될 수 있다.

이에 본 연구에서는 기존 연구들이 크게 다루지 않았던 '비전공자 사용자의 실제 경험'에 집중한다. 즉, 기초적인 프롬프트부터 결과 평가, 수정까지 실제 활용 전 과정을 아우르는 통합 활용 및 평가 프레임워크를 새롭게 설계하고 실증 데이터로 그 효과를 검증한다[10].

Ⅲ. 프레임워크 설계

3-1 프레임워크 개요

본 연구에서 제안하는 비전공자 중심 LLM 활용 프레임워크는 단순한 코드 생성 요청을 넘어, 문제 정의부터 결과 검증 및 오류 교정까지의 전 과정을 체계적으로 지원하도록 설계되었다. 알고리즘 문제를 해결하는 전 과정을 6단계로 구조화하고, 이를 반복하며 개선할 수 있도록 설계한 순환적 과정이 핵심이다[11]. 이러한 6단계 순환 프로세스의 전체 흐름은 그림 1과 같다.

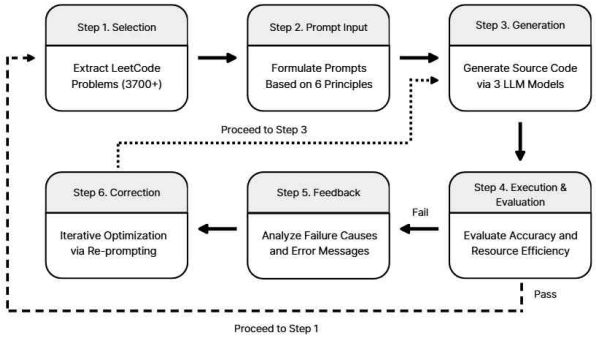


그림 1. 비전공자 중심 AI 협업 프레임워크 및 순환형 운영 프로세스

Fig. 1. Non-computer science user-centric AI collaboration framework and cyclic operation process

1) Step 1. 문제 선정(Selection)

LLM의 성능을 객관적으로 평가하고, 학습 데이터 오염으로 생길 수 있는 편향을 방지하기 위해 검증된 LeetCode 플랫폼에서 최신 알고리즘 문제를 난이도별로 선별한다. 이를 통해 모델 간 성능을 공정하게 비교할 수 있는 평가 환경을 구축한다.

2) Step 2. 프롬프트 입력(Prompt Input)

프로그래밍에 익숙하지 않은 비전공자의 한계를 보완하고자, 사전에 정의된 여섯 가지 프롬프트 구현 원칙, 즉 구현 언어 통제, 한글 주석 작성 의무화, 자원 효율성 최적화 지시, 구현 전 접근 방식 사전 분석, 코드 품질 및 예외 상황 고려, 함수 규격 준수를 적용하여 구조화된 질의문을 만들어 모델에 입력한다. 본 연구에서 6가지 원칙을 적용하여 활용한 실제 시스템 프롬프트 템플릿 지시문은 그림 2와 같다.

```
You will be provided with a LeetCode problem number and its description. Based on this information, derive the optimal solution adhering strictly to the rules below.

1. Implementation language: Write all code exclusively in Python 3, using standard libraries only.

2. Code documentation: Provide detailed Korean comments explaining each logical step and key variables within the code.

3. Performance optimization: Adopt an algorithm that minimizes time and space complexity within the given constraints.

4. Output format:
 - Briefly describe the chosen algorithmic approach and expected time complexity prior to code implementation.
 - Include an explicit analysis of both time and space complexity.

5. Code quality:
 - Apply Python type hinting.
 - Implement code ensuring robustness against all possible edge cases.

6. Implementation scope:
 - Write code strictly within the provided function signature.
```

그림 2. 제안 프레임워크의 표준 코드 생성 프롬프트

Fig. 2. Standard code generation prompt template of the proposed framework

3) Step 3. 코드 생성(Generation)

설계된 프롬프트에 따라 서로 다른 아키텍처와 실행 환경을 가진 여러 생성형 AI 도구(ChatGPT, Gemini, Copilot)를 활용해 목적에 부합하는 소스 코드를 생성한다.

4) Step 4. 실행 및 평가(Execution & Evaluation)

생성된 코드는 LeetCode 플랫폼의 자동 채점 시스템에 제출되며, 실시간으로 동적 검증을 거친다. 이 과정에서 코드의 정답률뿐 아니라 실행 속도, 메모리 사용량 등 자원 효율성도 수치로 측정된다.

5) Step 5. 피드백 수집(Feedback)

처음 생성된 코드에 오류가 있거나 최적화가 제대로 이루어지지 않은 경우, LeetCode 플랫폼이 반환하는 에러 메시지와 실패한 테스트 케이스 데이터를 체계적으로 기록해 수집한다. 이 단계에서는 사용자의 주관적 판단을 최대한 배제하고, 시스템이 제공하는 평가 결과만을 활용하는 것이 원칙이다.

6) Step 6. 오류 교정(Correction)

수집된 피드백 데이터를 다시 AI 모델에 입력함으로써, AI

가 초기 알고리즘 설계의 문제점이나 논리 오류를 직접 파악하고 코드를 수정할 수 있도록 유도한다. 이 자가 수정 과정은 무한 반복을 방지하기 위해 최대 10회까지만 수행하며, 이를 통해 최종 결과물의 신뢰성과 성능을 높인다.

3-2 AI 코드 생성 도구 선정 기준 및 대상

본 연구에서는 제안하는 프레임워크의 실효성을 검증하기 위해 도구 선정에 세 가지 기준을 적용하였다. 첫째, 대중성이 높고 기술적인 진입 장벽이 낮을 것, 둘째, 초기 도입 단계에서 비용 부담을 최소화할 것, 셋째, 다양한 구동 방식과 사용 환경을 지원할 것이다. 이 기준에 따라 ChatGPT 5.3, Google Gemini 3 Flash, GitHub Copilot(GPT-4o 기반)을 최종 실험 대상으로 선정하였다. 단, GitHub Copilot은 웹 기반 도구(ChatGPT, Gemini)와의 공정한 대화형 프롬프트 비교를 위해 실시간 코드 자동 완성 기능 대신 통합 개발 환경(IDE: integrated development environment) 내 대화형 인터페이스(Copilot Chat)만을 활용하도록 실험 조건을 통제하였다. 다만, IDE 내장 환경과 웹 브라우저 환경간의 문맥(Context) 인식 차이는 비전공자의 프롬프트 작성 및 초기 코드 생성 정밀도에 일부 영향을 미칠 수 있다.

또한 모든 모델의 코드 생성 언어는 Python 3로 통일하고, 매 문제마다 캐시가 초기화된 독립 세션을 사용하여 실험의 독립성과 신뢰성을 확보하였다.

3-3 평가 데이터셋 구축 및 문제 선정 전략

본 연구에서는 생성형 AI의 코드 생성 능력을 객관적으로 평가하기 위해 전 세계적으로 널리 활용되는 알고리즘 학습 플랫폼인 LeetCode를 평가 데이터셋 구축의 기반으로 선정하였다. LeetCode를 선택한 주요 이유는 세 가지다. 첫째, 데이터 오염을 막을 수 있다는 점이다. 기존 평가에 자주 사용된 HumanEval 등의 과거 데이터셋은 이미 AI 모델이 학습에 활용해 왔기 때문에[12], 본 연구에서는 LeetCode에서 3,700번대 이후의 최신 문항만을 추출하여 보다 객관적인 평가가 가능하도록 설계하였다. 둘째, 까다로운 예외 상황 검증이 가능하다는 점이다. LeetCode는 여러 개의 숨겨진 테스트 케이스를 모두 통과해야만 정답으로 인정하기 때문에 AI가 만든 코드의 미묘한 논리 오류까지도 효과적으로 걸러낼 수 있다. 셋째, 난이도 구분이 체계적이라는 점이다. Easy, Medium, Hard의 세 단계로 구분되어 있어 AI 모델의 문제 해결력 변화를 단계별로 분석하려는 본 연구에 가장 적합하다.

데이터 수집 기준일은 2026년 3월 20일로 설정하였으며, 수집된 문제들은 난이도별로 각각 20문제씩, Easy, Medium, Hard로 구분하여 총 60문항을 선정하였다. 이렇게 선정한 문제들은 3가지 AI 모델 모두에게 동일하게 적용하여 총 180개의 독립적인 테스트 데이터셋을 구축하였다. 이에 더해, 프레임워크의 실효성을 보다 정밀하게 입증하기 위해 초기 180문항 수집 이후 새롭게 출제된 최신 고난도(Hard) 10문항을 별도로 추출하여 프레임워크 적용 유무에 따른 성능 변화를 분석하는 제거 연구(Ablation Study) 환경을 별도로 추가 구성하였다.

3-4 생성 코드의 평가 지표

본 연구에서는 생성된 코드의 성능을 객관적이고 정량적으로 평가하기 위해 다음 세 가지 평가 지표를 정의한다.

1) 기능적 정답률 및 Zero-Shot Pass@1

AI 모델의 순수한 문제 해결 능력을 평가하는 핵심 지표로 '단일 시도 정답률(Pass@1)'을 사용한다. 힌트나 코드 수정 없이, AI가 처음 내놓은 코드 하나만을 LeetCode에 입력해 모든 테스트 케이스를 한 번에 통과(Accepted)해야만 정답으로 인정하는 엄격한 평가 방식을 적용한다. 최초 시도에서 실패한 코드의 경우에는 에러 피드백을 제공하며 최대 10회까지 코드를 수정하도록 요청하는 자가 수정(Self-Repair) 능력 검증도 병행하였으나, 해당 결과는 참고 데이터로만 활용하며 Pass@1 산정에는 포함하지 않는다. 단, 자가 수정(Self-Repair)이 모든 코드 생성 오류를 해결하는 만능 수단은 아님을 유의해야 한다[13].

2) 자원 효율성(실행 효율 및 메모리 효율)

코드의 품질은 정답 여부만으로 판단할 수 없으므로, 실행 효율(Runtime Beats)과 메모리 효율(Memory Beats)을 LeetCode 플랫폼의 채점 결과 화면에 나타난 상대적 백분위(Beats %) 수치로 측정하여 수집하였다. 실제로 최근 연구에서도 구조화된 프롬프트가 LLM의 코드 생성 성능과 자원 효율성을 실질적으로 향상시킨다는 점이 실증적으로 확인된 바 있다[14].

3) 오류 유형 분류

Pass@1 기준을 충족하지 못해 오답으로 판정된 코드는 실제 발생 에러 로그를 기반으로 '구문 및 실행 오류', '논리 오류', '자원 한계 초과'의 세 가지 핵심 유형으로 분류하여 분석하였다. 특히 구현 전 접근 방식 사전 분석 원칙은 연쇄 추론(Chain-of-Thought) 기법을 자연스럽게 이끌어내어 논리 전개의 구조적 오류를 사전에 방지하는 데 효과적임이 보고된 바 있다[15]. 또한, 구조화된 프롬프트가 LLM의 판정 정확도와 결과 신뢰도를 높일 수 있다는 선행 연구 결과와도 부합한다[16].

Ⅳ. 제안 프레임워크의 적용 가능성 및 실효성 검증

본 장에서는 Ⅲ장에서 설계한 비전공자 중심 LLM 기반 코

드 생성 활용 및 평가 프레임워크를 실제 코드 생성 과제에 적용하여 그 실효성을 검증한다. 이를 위해 ChatGPT 5.3, Gemini 3 Flash, GitHub Copilot을 대상으로 동일한 문제와 절차를 적용하고, 난이도별 정답률, 실행 효율, 메모리 효율을 비교하였다. 이후 가장 안정적인 결과를 보인 모델을 대상으로 일반 요청 방식과 프레임워크 적용 방식을 비교함으로써, 제안 프레임워크가 코드 생성 결과의 정확성과 효율성을 개선하는지 확인하였다.

4-1 프레임워크 적용 조건에서의 도구별 안정성 비교

본 절에서는 제안 프레임워크를 동일하게 적용한 조건에서 ChatGPT 5.3, Gemini 3 Flash, GitHub Copilot의 문제 해결 안정성을 비교한다. 성능평가에는 LeetCode에서 최근 추가된 3,700번대 이후의 알고리즘 문제 180개를 사용하였으며, 세 도구 모두에 동일한 프레임워크와 평가 절차를 적용하였다. 표 2는 1회 시도 정답률(Zero-Shot Pass@1) 및 자원 효율성을 통합하여 제시한 것이며, 그림 3은 난이도 및 AI 도구별 1회 시도 정답률을 비교하여 나타낸다.

표 2. 프레임워크 적용 조건에서의 도구별 Pass@1 및 자원 효율성

Table 2. Pass@1 and resource efficiency by AI tool under framework application

Model	Zero-Shot Pass@1			Total	Runtime Beats	Memory Beats
	Easy	Medium	Hard	Overall	(Avg.)	(Avg.)
ChatGPT 5.3	100%	65%	50%	71.7%	63.3%	61.0%
Gemini 3 Flash	100%	85%	95%	93.3%	65.3%	64.5%
GitHub Copilot	80%	55%	20%	51.7%	56.6%	59.4%

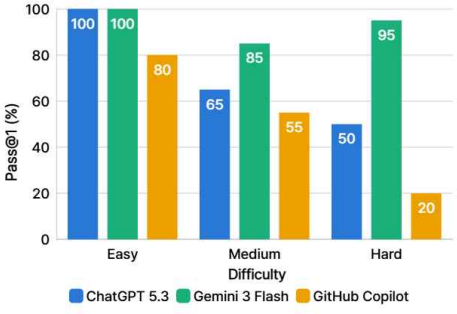


그림 3. 프레임워크 적용 조건에서 난이도 및 AI 도구별 1회 시도 정답률 비교

Fig. 3. Zero-Shot Pass@1 by AI tool and difficulty under framework application

동일한 프레임워크 적용 조건에서 Easy 난이도에서는 세 도구 모두 비교적 높은 정답률을 기록한 반면, Hard 난이도로 갈수록 도구 간 격차가 크게 벌어졌다. Gemini 3 Flash는 전체 정답률 93.3%를 기록하였으며, 실행 효율(65.3%)과 메모리 효율(64.5%)에서도 가장 높은 수치를 보여 정답률, 실행 효율, 메모리 효율 측면 모두 가장 안정적인 결과를 보였다. 이러한 결과를 바탕으로 본 연구에서는 Gemini 3 Flash를 일반 요청 방식과 제안 프레임워크 적용 방식을 비교하기 위한 제거 연구(Ablation Study)의 대표 모델로 선정하였다.

4-2 프레임워크 적용 조건에서의 잔존 오류 위험 요인

본 절에서는 프레임워크 적용 이후에도 남아 있는 오류 유형을 분석하고, 비전공자가 지속적으로 점검해야 할 위험 요인을 도출한다. 분석 대상은 4-1절의 180개 데이터 중 1회 시도에서 실패한 총 50개 데이터(ChatGPT 17건, Gemini 3 Flash 4건, GitHub Copilot 29건)이며, 이를 '구문 및 실행 오류', '논리 오류', '자원 한계 초과'의 세 가지 유형으로 분류하였다. 결과는 표 3과 같다.

표 3. 프레임워크 적용 조건에서의 1회 시도 실패 데이터 오류 유형

Table 3. Error type distribution of first-attempt failures under framework application

Model	Total Failures	Syntax & Runtime Error	Logic Error	Resource Limit Exceeded
ChatGPT 5.3	17 cases	0 (0%)	12 (70.6%)	5 (29.4%)
Gemini 3 Flash	4 cases	0 (0%)	2 (50.0%)	2 (50.0%)
GitHub Copilot	29 cases	3 (10.3%)	13 (44.8%)	13 (44.8%)

주목할 점은 프레임워크 적용 이후에도 논리 오류와 자원 한계 초과가 모든 도구에서 여전히 나타났다는 점이다. Gemini 3 Flash의 경우 전체 실패 사례는 4건으로 가장 적었지만, 이 중 2건은 논리 오류, 2건은 자원 한계 초과로 분류되었다. 이는 프레임워크를 적용하더라도 생성된 코드를 그대로 신뢰하기보다는, 실행 결과와 오류 유형을 지속적으로 점검하는 절차가 필요함을 보여준다. 실제로 최근 연구에서도 최상위 AI 모델조차 복잡한 예외와 경계 상황에서는 전문가 수준의 완성도를 안정적으로 보장하지 못하는 경우가 있음이 확인된 바 있다[17]. 따라서 이러한 잔존 오류는 다음 절에서 프레임워크 적용 전후의 오류 통제 효과를 추가로 검증할 필요성을 뒷받침한다.

4-3 제안 프레임워크 적용 전후 실효성 검증(Ablation Study)

본 절에서는 4-1절에서 선정한 Gemini 3 Flash를 대상으로 일반 요청 방식과 제안 프레임워크 적용 방식의 성능 차이를 비교하여, 프레임워크 적용 여부에 따른 효과를 검증하였다. 본 성능 평가에서는 데이터 오염 가능성을 최소화하기 위

해 평가 시점 기준으로 가장 최근에 추가된 LeetCode 3,900번대 이후의 Hard 난이도 10문항을 최신순 내림차순으로 선별하여 사용하였다. 해당 문항들은 동적 계획법(Dynamic Programming), 복합 그래프 탐색 등 정밀한 논리 추론과 예외 처리가 요구되는 최신 알고리즘 문제들로 구성되어 제안 프레임워크의 실효성을 검증하기에 적합하다. 일반 요청 조건에서는 별도의 구조화된 지침 없이 문제 해결만을 요청하였고, 프레임워크 적용 조건에서는 Ⅲ장에서 제시한 6대 프롬프트 원칙과 실행·평가·피드백·교정 절차를 동일하게 적용하였다. 결과는 표 4 및 그림 4와 같다.

표 4. 제안 프레임워크 적용 전후 제거 연구 결과
Table 4. Ablation study results before and after framework application

Metric	Baseline (No Framework)	With Proposed Framework	Change
Zero-Shot Pass@1	40% (4 items)	80% (8 items)	+40 pp
Syntax & Runtime Error	3 cases	0 cases	−3 cases
Logic Error	2 cases	1 case	−1 case
Resource Limit Exceeded	1 case	1 case	No change
Avg. Runtime Efficiency (Runtime Beats)	38.7%	50.5%	+11.8 pp
Avg. Memory Efficiency (Memory Beats)	40.1%	59.6%	+19.5 pp

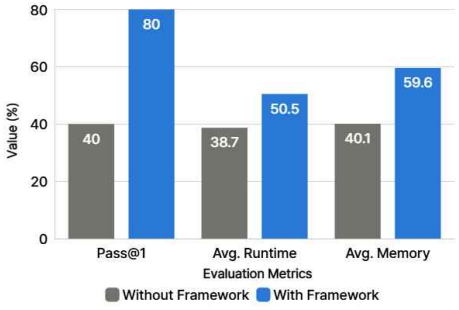


그림 4. 프레임워크 적용 전후 정답률 및 자원 효율성 변화
Fig. 4. Performance changes before and after framework application

분석 결과, 프레임워크 적용 후 주요 지표에서 개선이 확인되었다.

첫째, 1회 시도 정답률(Zero-Shot Pass@1)은 일반 요청 방식에서 40%였으나, 프레임워크 적용 후 80%로 향상되었다. 이는 동일한 모델을 사용하더라도 문제 해결 절차를 구조화하는 방식에 따라 정답률이 달라질 수 있음을 보여준다.

둘째, 오류 발생 양상에서도 변화가 나타났다. 일반 요청 방식에서는 총 6건의 실패가 발생하였으나, 프레임워크 적용 후에는 실패 건수가 2건으로 감소하였다. 특히 비전공자가 원인을 파악하고 대응하기 어려운 구문 및 실행 오류가 3건에서 0건으로 줄어든 점은 주목할 만하다. 이는 한글 주석 의무화, 예외 상황 고려, 함수 규격 준수 등의 프롬프트 설계 원칙이 오류 발생 가능성을 낮추는 데 기여했음을 시사한다.

셋째, 자원 효율성 측면에서도 개선이 확인되었다. 평균 실행 효율(Runtime Beats)은 38.7%에서 50.5%로 11.8%p, 평균 메모리 효율(Memory Beats)은 40.1%에서 59.6%로 19.5%p 향상되었다. 이는 프레임워크에 포함된 자원 효율성 최적화 지시와 구현 전 접근 방식 사전 분석 원칙이 정답 여부뿐 아니라 코드의 실행 품질 개선에도 기여할 수 있음을 보여준다.

이상의 결과는 제안 프레임워크가 정답률 향상, 오류 통제, 자원 효율성 개선에 긍정적인 영향을 미쳤음을 보여준다. 특히 동일한 모델을 사용하더라도 문제 해결 절차를 구조화함으로써 코드 생성 결과의 정확성과 효율성을 개선할 수 있음을 확인하였다.

4-4 종합 논의

앞선 분석 결과를 종합하면, 제안 프레임워크의 적용은 비전공자의 LLM 기반 코드 생성 활용에서 다음과 같은 시사점을 갖는다.

첫째, 비전공자 환경에서는 AI 도구 자체의 인지도나 편의성보다, 생성된 결과물을 구조화된 절차로 검증하는 과정이 더 중요하다. 동일한 모델인 Gemini 3 Flash를 사용하더라도 일반 요청 방식과 프레임워크 적용 방식 사이에서 1회 시도 정답률이 40%에서 80%로 향상된 결과는 이를 뒷받침한다.

둘째, AI 생성 코드의 품질은 정답 여부만으로 판단하기 어렵다. 본 연구에서 프레임워크 적용 후 평균 실행 효율과 메모리 효율이 함께 개선된 것은, 비전공자 대상 코드 생성 평가에서 자원 효율성 지표를 함께 고려할 필요가 있음을 보여준다.

셋째, 프레임워크 적용 후에도 논리 오류와 자원 한계 초과는 일부 남아 있었다. 따라서 비전공자가 LLM 기반 코드 생성 도구를 활용할 때에는 프레임워크를 따르더라도 최종 결과물에 대한 최소한의 비판적 검토와 실행 결과 확인을 병행해야 한다.

Ⅴ. 결 론

본 연구에서는 비전공자가 LLM 기반 코드 생성 도구를 보다 안전하고 효율적으로 활용할 수 있도록 문제 선정부터 오류 교정까지 아우르는 6단계 순환 프레임워크를 제안하였다. LeetCode 기반 성능 평가를 통해 1회 시도 정답률 향상, 구문 및 실행 오류 감소, 실행 효율과 메모리 효율 개선을 확인하였으며, 이를 통해 본 제안 프레임워크가 코드 생성 결과의

정확성과 효율성을 개선하는 데 효과적임을 검증하였다. 특히 본 연구는 코딩 지식이 부족한 일반 사용자의 관점에서 실무 적용이 가능한 표준화된 활용 가이드라인을 제시했다는 점에서 실무적 의의를 갖는다.

감사의 글

본 연구는 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지역지능화혁신인재양성사업(IITP-2026-RS-2020-II201462)의 지원을 받아 수행되었음.

참고문헌

[1] E. Elshan, B. Binzer, and T. J. Winkler, “From Software Users to Software Creators: An Exploration of the Core Characteristics of the Citizen Developer Role and the Related Re- and Upskilling Programs,” *Business & Information Systems Engineering*, Vol. 67, No. 1, pp. 31-53, February 2025. https://doi.org/10.1007/s12599-024-00915-1

[2] P. Haindl and G. Weinberger, “Does ChatGPT Help Novice Programmers Write Better Code? Results from Static Code Analysis,” *IEEE Access*, Vol. 12, pp. 114146-114156, August 2024. https://doi.org/10.1109/ACCESS.2024.3445432

[3] A. Bayram, G. G. M. Dalveren, and M. Derawi, “Comparative Analysis of AI Models for Python Code Generation: A HumanEval Benchmark Study,” *Applied Sciences*, Vol. 15, No. 18, 9907, September 2025. https://doi.org/10.3390/app15189907

[4] P. Vaithilingam, T. Zhang, and E. L. Glassman, “Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models,” in *Proceedings of the CHI Conference on Human Factors in Computing Systems Extended Abstracts*, New Orleans: LA, pp. 1-7, April 2022. https://doi.org/10.1145/3491101.3519665

[5] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, “Large Language Models for Software Engineering: Survey and Open Problems,” arXiv:2310.03533, November 2024. https://doi.org/10.48550/arXiv.2310.03533

[6] P. Denny, V. Kumar, and N. Giacaman, “Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language,” in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, Toronto, Canada, pp. 1136-1142, March 2023. https://doi.org/10.1145/3545945.3569823

[7] I. D. Fagadau, L. Mariani, D. Micucci, and O. Riganelli, “Analyzing Prompt Influence on Automated Method Generation: An Empirical Study with Copilot,” in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, Lisbon, Portugal, pp. 24-34, April 2024. https://doi.org/10.1145/3643916.3644409

[8] Y. Chen, Y. Li, B. Ding, and J. Zhou, “On the Design and Analysis of LLM-Based Algorithms,” arXiv:2407.14788v2, September 2024.

[9] D. Huang, Y. Qing, W. Shang, H. Cui, and J. M. Zhang, “EffiBench: Benchmarking the Efficiency of Automatically Generated Code,” in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, Vancouver, Canada, pp. 11506-11544, December 2024. https://doi.org/10.48550/arXiv.2402.02037

[10] C. Kang, Y. Kim, C. Yu, S. Yoon, Y. Lee, and K. Yu, “Personalized Museum Docent System Based on Multimodal Retrieval-Augmented Generation and Large Language Models,” *Journal of Digital Contents Society*, Vol. 27, No. 2, pp. 473-484, 2026. https://doi.org/10.9728/dcs.2026.27.2.473

[11] H. Chun, “Study on Prompting in Generative AI: A Creative Writing Tool for the Digital Era,” *Journal of Digital Contents Society*, Vol. 26, No. 2, pp. 275-285, 2025. https://doi.org/10.9728/dcs.2025.26.2.275

[12] N. Jain, K. Han, A. Gu, W. Li, F. Yan, T. Zhang, ... and I. Stoica, “LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code,” arXiv:2403.07974, March 2024. https://doi.org/10.48550/arXiv.2403.07974

[13] T. X. Olausson, J. P. Inala, C. Wang, J. Gao, and A. Solar-Lezama, “Is Self-Repair a Silver Bullet for Code Generation?,” arXiv:2306.09896, June 2023. https://doi.org/10.48550/arXiv.2306.09896

[14] A. Midolo, A. Giagnorio, F. Zampetti, R. Tufano, G. Bavota, and M. Di Penta, “Guidelines to Prompt Large Language Models for Code Generation: An Empirical Characterization,” arXiv:2601.13118, January 2026. https://doi.org/10.48550/arXiv.2601.13118

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, ... and D. Zhou, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” in *Proceedings of the 36th International Conference on Neural Information Processing Systems,* New Orleans: LA, pp. 24824-24837, December 2022.

[16] Y. Choi and T. Oh, “Comparative Study of Prompt

Engineering Strategies for LLM-Based Automated Grading of Career Sentence Completion Test," *Journal of Digital Contents Society*, Vol. 27, No. 4, pp. 1061-1068, 2026. https://doi.org/10.9728/dcs.2026.27.4.1061

[17] Z. Zheng, Z. Cheng, Z. Shen, S. Zhou, K. Liu, H. He, ... and S. Xie, "LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming?," arXiv:2506.11928, June 2025. https://doi.org/10.48550/arXiv.2506.11928



정인철(Incheol Jeong)

2026년 : 충북대학교 전자정보공학과 공학석사

2021년~현 재: 주식회사 에피젠

※관심분야 : AI 기반 코드 생성(AI-driven Code Generation), 알고리즘 성능 평가(Algorithm Performance Evaluation)



김태홍(Taehong Kim)

2012년 : KAIST 전산학 박사

2012년~2014년: 삼성전자 책임연구원

2014년~2016년: 한국전자통신연구원 선임연구원

2016년~현 재: 충북대학교 정보통신공학부 정교수

※관심분야 : 연합학습(Federated Learning), 엣지컴퓨팅(Edge Computing), 사물인터넷(Internet of Things)



유재수(Jaesoo Yoo)

1995년 : KAIST 전산학과 (공학박사)

1995년~1996년: 목포대학교 전산통계학과 전임 강사

2009년~2010년: California State University 방문교수

2019년~2020년: California State University 방문교수

1996년~현 재: 충북대학교 정보통신공학부 정교수

※관심분야 : 데이터베이스 시스템(Database Systems), 빅데이터(Big Data), 그래프 데이터 처리(Graph Data Processing), 추천 시스템(Recommendation Systems)