JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 31 No. 11, pp. 117-129, November 2026
https://doi.org/10.9708/jksci.2026.31.11.117

# An Exploratory Comparison of Vibe-Coding Prompt Conditions

Chul-Hee Lee*, Sang-Yong Choi*

*Professor, Dept. of Cloud Security, Korea Polytechnics, Daejeon, Korea

## [Abstract]

This study explores how prompting conditions relate to code quality in vibe coding. We evaluated 72 outputs across six tasks and four compound prompt conditions. Omnibus differences in test pass rates and security warning scores remained significant in sensitivity analyses. Specification asymmetry, evaluator compatibility, and test overlap precluded attribution to individual strategies; complexity and code smell findings were sensitive to metric composition. A separate factorial comparison held functional requirements, technology stack, and interfaces constant while varying prompt format, additional security implementation guidance, and public example tests. The 288 outputs across 12 tasks were evaluated on held-out scenarios. All 2,304 scenarios executed, with an overall pass rate of 98.35%. Structured formatting and public examples each showed a marginal difference of +3.30 percentage points, but neither was significant under task-level exact tests with Holm adjustment. Only one security warning was detected. High pass rates and differences concentrated in few tasks limited assessment of additional effects; nonsignificance was not interpreted as equivalence. These findings highlight the need to distinguish compound-condition differences from controlled factor comparisons when interpreting generated code quality. Conclusions are limited to the tested model, tasks, and operational definitions.

▸ **Key words**: Vibe Coding, LLM, Prompt Engineering, Code Quality, Security Vulnerability, Code Smell

## [요 약]

자연어 프롬프트와 대규모 언어 모델을 이용하는 바이브 코딩에서 프롬프트 조건에 따른 생성 코드의 품질을 탐색적으로 비교하였다. 6과제에 네 가지 복합 프롬프트 조건을 적용하여 생성한 72개 산출물을 비교하였다. 테스트 통과율과 보안 경고 점수의 전체 조건 차이는 민감도 분석에서도 유의하게 유지되었다. 다만 조건 간 정보 차이, 평가 호환성 및 테스트 중복을 고려하여 특정 전략의 독립 효과로 해석하지 않았으며, 복잡도·코드 스멜 결과는 지표 구성에 민감하였다. 또한 동일한 기능·기술 스택·인터페이스 명세 아래 프롬프트 형식, 추가 보안 구현 지침, 공개 예시 테스트의 세 요인을 독립적으로 조작하였다. 12과제에서 생성한 288개 산출물에 미제공 평가를 적용한 결과, 2,304개 시나리오가 모두 실행되었고 평균 통과율은 98.35%였다. 형식 구조화와 공개 예시 제공의 주변 평균 차이는 각각 +3.30%p였으나 과제 단위 정확 검정과 Holm 보정에서 유의하지 않았다. 보안 경고는 한 개로 희소하였다. 높은 통과율과 소수 과제에 집중된 차이로 추가 효과의 판단에 제약이 있었으며, 비유의성을 동등성의 근거로 해석하지 않았다. 본 연구는 복합 조건의 차이와 통제된 요인 비교를 구분하여 생성 코드 품질을 해석할 필요성을 보여준다.

▸ **주제어**: 바이브 코딩, 대규모 언어 모델, 프롬프트 엔지니어링, 코드 품질, 보안 취약점, 코드 스멜

• First Author: Chul-Hee Lee, Corresponding Author: Sang-Yong Choi
*Chul-Hee Lee (cjfclrchr@gmail.com), Dept. of Cloud Security, Korea Polytechnics
*Sang-Yong Choi (csyong95@gmail.com), Dept. of Cloud Security, Korea Polytechnics
• Received: 2026. 08. 04, Revised: 2026. 10. 14, Accepted: 2026. 10. 26.

Copyright © 2026 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945
www.kci.go.kr

## I. Introduction

생성형 AI와 대규모 언어 모델(LLM)의 발전은 소프트웨어 개발 패러다임에도 큰 변화를 가져오고 있다. 그 중심에 있는 바이브 코딩(Vibe Coding)은 2025년 Karpathy가 처음 제안한 용어로, 개발자가 자연어 프롬프트만으로 코드 생성·수정·기능 확장 과정을 LLM에 위임하는 개발 방식을 의미한다[1]. 기존의 AI 보조 개발이 자동완성·함수 추천·오류 수정 제안 등 보조적 역할에 초점을 두었다면, 바이브 코딩은 요구사항 기술부터 구현까지의 상당 부분을 자연어 상호작용에 의존한다는 점에서 차별화된다. 실제로 교육, 프로토타이핑, 에이전트 기반 개발 등 다양한 맥락에서 활용 가능성이 논의되고 있다[2].

그러나 바이브 코딩의 확산과 함께 생성 코드의 품질 및 신뢰성에 대한 우려도 제기되고 있다. 바이브 코딩 환경에서는 사용자가 LLM이 생성한 코드를 충분히 이해하거나 검토하지 않은 상태에서 결과물을 수용할 가능성이 있으며, 이 경우 기능적으로는 실행 가능하지만 보안성, 유지보수성, 아키텍처 일관성이 부족한 코드가 생성될 수 있다. 특히 LLM이 생성한 코드는 프롬프트에 명시되지 않은 보안·검증 관련 속성을 일관되게 반영하지 못할 수 있으며, 프롬프트 구성 방식에 따라 결함 유발률과 보안 취약점 발생 가능성이 달라질 수 있음이 보고되었다[3], [4].

기존 연구들은 바이브 코딩의 개념적 특성, 실무 적용 경험, 교육적 활용 가능성 등을 논의해 왔으며[1-3], LLM 기반 코드 생성의 기능적 성능을 평가하는 연구도 보고 되었다[5]. 그러나 프롬프트의 구조화 수준 차이가 바이브 코딩 결과물의 품질에 미치는 영향을 통제된 실험 설계에 기반하여 정량적으로 비교한 연구는 아직 제한적이다. 특히 단순한 기능 구현 성공 여부뿐 아니라 보안 취약점, 코드 복잡도, 코드 스멜과 같이 정적 분석 및 테스트 기반으로 정량화 가능한 품질 속성을 통합적으로 비교한 연구는 충분히 다루어지지 않았다.

한편 기존 코드 생성 평가에서는 HumanEval, MBPP와 같은 벤치마크 및 pass@k 지표가 널리 활용되어 왔다[5]. 이러한 평가는 생성 코드가 사전 정의된 테스트 케이스를 통과하는지를 기준으로 기능 정확성을 측정한다는 점에서 유용하다. 그러나 바이브 코딩은 단일 함수 수준의 프로그램 생성이 아니라 자연어 요구사항을 바탕으로 소규모 애플리케이션 또는 기능 단위를 구성하는 방식이므로, 기능 정확성 중심의 평가만으로는 생성 결과물의 실제 품질을 충분히 설명하기 어렵다. 따라서 바이브 코딩의 한계를 객관적으로 규명하기 위해서는 기능성뿐 아니라 보안성, 유지보수성, 복잡도를 포함하는 다차원 평가가 필요하다.

이에 본 연구는 프롬프트 조건에 따른 바이브 코딩 결과물의 품질 차이를 정량적으로 비교하는 것을 목적으로 한다. 이를 위해 복잡도 수준(Level 1-3)이 서로 다른 6개의 소프트웨어 개발 과제를 설계하고, 네 가지 프롬프트 조건(C1: Pure Vibe, C2: Iterative Vibe, C3: Structured Prompt, C4: Test-Driven Vibe)을 적용한다. 생성된 결과물은 테스트 통과율, 정적 분석 기반 보안 경고 가중 점수, 사이클로매틱 복잡도, 코드 스멜 수의 네 가지 품질 메트릭을 기준으로 비교·분석한다.

다만 조건 간 기술 스택, 보안 요구사항, 출력 형식과 호출 수가 함께 달라지므로, C1~C4 비교는 구조화 형식 자체의 독립적 인과효과를 검증하는 통제 실험이 아닌 복합 프롬프트 조건의 탐색적 비교로 해석한다.

또한 동일한 기능·기술 스택·인터페이스 명세 아래 프롬프트 형식(F), 추가 보안 구현 지침(S) 및 공개 예시 테스트(T)의 세 요인을 독립적으로 조작하여, 각 요인에 따른 미제공 시나리오 통과율 및 보안 경고 점수의 차이를 살펴보았다. 이 요인 비교와 C1~C4의 복합 조건 비교는 과제 명세와 평가가 다르므로 각각 분석하였다.

본 연구의 핵심 연구 질문(Research Question, RQ)은 다음과 같다.

* RQ1: 프롬프트 조건(C1~C4)에 따라 바이브 코딩 결과물의 테스트 통과율·보안 경고 점수·코드 복잡도·코드 스멜수에 어떤 차이가 나타나는가?
* RQ2: 과제 복잡도(Level 1~3)에 따라 프롬프트 조건 간 품질 격차는 어떠한 양상으로 나타나는가?
* RQ3: C1~C4 비교 결과와 해석의 한계는 무엇이며, 공통 명세에서 F·S·T에 따른 미제공 시나리오 통과율과 보안 경고 점수는 어떻게 달라지는가?

본 논문의 구성은 다음과 같다. II장에서는 바이브 코딩의 정의와 관련 선행 연구 및 기존 연구의 한계를 논의한다. III장에서는 실험 과제 설계, 프롬프트 조건 정의, 측정 메트릭 및 실험 환경을 상세히 기술한다. IV장에서는 조건별 실험 결과를 제시하고 비교 분석을 수행한다. V장에서는 연구 결과를 요약하고 기여와 한계를 논의하며 향후 연구 방향을 제시한다.

www.kci.go.kr

## II. Related Work and Background

### 2.1 LLM-Based Code Generation and Quality Evaluation

LLM 기반 코드 생성의 기능적 성능은 HumanEval과 pass@k 등의 벤치마크를 중심으로 평가되어 왔으나[5], 이러한 평가는 주로 함수 단위의 문제 해결과 테스트 통과 여부에 초점을 두므로 실제 소프트웨어 개발에서 요구되는 보안성, 코드 복잡도 및 유지보수성을 종합적으로 설명하는 데 한계가 있다.

생성형 코드 도구가 산출한 코드에서는 보안에 취약한 구현 패턴이 나타날 수 있으며, 보안 요구사항이 명시되지 않은 상황에서 이러한 위험이 충분히 반영되지 않을 수 있음이 보고되었다[3]. 반면 제한된 C 프로그래밍 과제의 사용자 연구에서는 보안 위험의 큰 증가가 관찰되지 않았다[4]. 따라서 기능 정확성과 별개로 생성 코드 자체의 보안성을 평가하고, 과제 및 사용 맥락에 맞게 해석할 필요가 있다.

기계학습 문제를 통해 LLM의 코딩 능력을 평가하고[6], 생성 코드의 문법적 정확성과 활용 가능성을 평가하려는 연구가 수행되었다[7]. 한편, 자동 프로그램 수리 연구에서는 LLM을 활용하여 코드 결함을 탐지·수정하는 가능성이 제시되었으며[8], 이는 LLM 기반 코드 연구의 범위가 단순 코드 생성에서 생성 결과의 오류 수정과 품질 개선으로 확장되고 있음을 보여준다.

그러나 기존 연구는 기능 정확성, 개발 효율성, 보안성 또는 자동 수정 가능성 가운데 일부 측면을 개별적으로 다루는 경우가 많으며, 본 연구는 프롬프트 조건별 테스트 통과율, 보안 경고, 코드 복잡도 및 유지보수성 관련 코드 스멜을 함께 비교한다. 이때 조건의 정보량 차이와 각 지표의 측정 범위를 구분한다.

### 2.2 Prompt Engineering Techniques

프롬프트 엔지니어링은 LLM이 의도한 출력을 생성하도록 역할, 제약 조건, 출력 형식 등의 입력 요소를 구조화하는 실천이다. 반복 가능한 프롬프트 설계 전략은 패턴 카탈로그로 체계화되어 왔으며[9], 동일한 기능 요청에서도 이러한 구성 요소와 테스트 조건의 포함 여부에 따라 생성 결과가 달라질 수 있다.

국내에서도 LLM 기반 코드 생성을 위한 프롬프트 조합 방법[10]과 프로그래밍 학습 환경에서의 프롬프트 프로토콜 설계[11]가 논의되었다. 이러한 연구는 프롬프트를 모델의 출력 품질을 조정하는 설계 요소로 다룬다는 점에서 본 연구의 조건 구성에 이론적 근거를 제공한다.

한편, 추론 과정을 단계적으로 유도하는 CoT(Chain-of-Thought) 프롬프팅이 제안되었으나[12], 본 연구는 요구사항과 테스트 명세의 구조화 수준을 비교하는 데 초점을 두어 이를 별도 조건으로 설정하지 않았다. 또한 LLM 생성 코드의 보안 취약점을 체계적으로 평가하기 위한 데이터셋과 분석 체계가 제안되었으며[13], 본 연구에서는 기능 요구사항뿐 아니라 보안 요구사항의 포함 여부를 프롬프트 구조화의 주요 요소로 고려하였다.

### 2.3 Code Quality Metrics

본 연구에서는 생성 코드의 품질을 기능 정확성, 코드 복잡도 및 코드 스멜을 중심으로 평가하며, 보안 취약점 평가는 다음 절에서 별도로 기술한다.

#### 2.3.1 Functional Correctness

기능 정확성은 생성된 코드가 주어진 요구사항을 올바르게 수행하는 정도를 의미한다. 코드 생성 연구에서는 일반적으로 생성 결과 중 하나 이상이 테스트를 통과할 확률인 pass@k를 활용하며[5], 본 연구에서는 각 산출물이 통과한 테스트의 비율을 사용하므로 pass@1과 구분한다. 테스트 구성과 적용 절차는 III장 3.5.1에서 제시한다.

#### 2.3.2 Code Complexity

사이클로매틱 복잡도(Cyclomatic Complexity)는 프로그램의 제어 흐름 그래프에서 선형 독립 경로의 수를 측정하며, 값이 높을수록 독립적인 테스트 경로 수가 증가한다[14]. 본 연구에서 적용한 복잡도 등급과 세부 측정 절차는 III장 3.5.3에서 설명한다.

#### 2.3.3 Code Smell

코드 스멜(Code Smell)은 프로그램이 즉각적인 오류를 발생시키지는 않지만, 장기적으로 유지보수성과 확장성을 저하시킬 수 있는 구조적 징후를 의미한다[15]. 본 연구에서는 Pylint 기반 코드 스멜 수를 유지보수성의 대리 지표로 사용하며, 세부 측정 기준은 III장 3.5.4에서 제시한다.

### 2.4 Security Vulnerability Evaluation Framework

생성형 AI는 사용자가 명시적으로 보안 요구사항을 제시하지 않을 경우, 실행 가능하지만 취약한 코드를 생성할 수 있다. 특히 웹 애플리케이션, 인증 기능, 데이터베이스 연동, 파일 업로드, 명령어 실행과 같은 과제에서 보안 취약점이 빈번하게 발생하는 것이 확인되었다[16].

www.kci.go.kr

Table 1. CWE-Based Weakness Types and Research-Defined Severity Categories

| Severity | CWE | Vulnerability Type | Description |
|---|---|---|---|
| Critical | CWE-89 | SQL Injection | A vulnerability in which user input is inserted into an SQL query without proper validation or parameterization |
| Critical | CWE-798 | Hard-coded Credentials | A vulnerability in which sensitive information, such as passwords, API keys, or tokens, is hard-coded in the source code |
| Critical | CWE-327 | Broken or Risky Cryptographic Algorithm | A vulnerability caused by the use of an insecure cryptographic algorithm |
| High | CWE-22 | Path Traversal | A vulnerability that allows access to unauthorized file paths through user input |
| High | CWE-78 | OS Command Injection | A vulnerability in which user input can be exploited to execute operating-system commands |
| High | CWE-79 | Cross-Site Scripting | A vulnerability in which malicious scripts are executed in a browser because user input is not properly escaped |
| Medium | CWE-20 | Improper Input Validation | A vulnerability caused by insufficient input validation that fails to handle exceptional cases or malicious input appropriately |
| Medium | CWE-306 | Missing Authentication for Critical Function | A vulnerability in which an authentication procedure is omitted from a critical function |
| Medium | CWE-862 | Missing Authorization | A vulnerability in which the permissions of an authenticated user are not verified |

본 연구에서는 OWASP Top 10과 CWE(Common Weakness Enumeration)를 보안 평가의 배경으로 활용한다. OWASP Top 10은 웹 애플리케이션에서 위험도가 높은 보안 위협을 정리한 기준으로, 접근 제어 실패, 암호화 실패, 인젝션, 보안 설정 오류, 인증 실패 등을 포함하며[17], 본 연구의 실험 과제는 이와 관련된 보안 요소를 포함하도록 설계하였다. CWE는 코드 수준에서 발생하는 구체적인 결함 유형을 식별하는 표준 분류 체계이다[18]. 본 연구에서는 Bandit 경고의 CWE를 Table 1의 연구자 정의 범주에 매핑한다. 이 범주는 CWE 자체의 공식 심각도와 구분한다.

보안 경고 탐지에는 Python 정적 분석 도구인 Bandit을 활용한다. Bandit은 하드코딩된 비밀번호, 안전하지 않은 함수 호출, 취약한 암호화 알고리즘 사용 등을 탐지하는 데 사용된다[19]. 정적 분석은 오탐(false positive) 및 미탐(false negative) 가능성을 가지므로, 경고를 실제 취약점수와 구분하며 세부 집계 기준과 한계는 III장 실험 설계에서 기술한다.

## III. Experimental Design

### 3.1 Experimental Overview

프롬프트 조건에 따른 바이브 코딩 결과물의 품질 차이를 비교하였다. 비교 조건은 프롬프트 조건(C1-C4)이며, 평가 지표는 테스트 통과율(M1), 보안 경고 가중 점수(M2), 코드 복잡도(M3), 코드 스멜 수(M4)의 네 가지 품질 메트릭이다. 과제 복잡도(Level 1-3)는 추가 분석 요인으로 설정하여, 복잡도 수준에 따라 프롬프트 조건간 품질 차이가 어떻게 변화하는지를 분석하였다.

3.1~3.6절은 6과제·C1~C4 조건의 비교를, 3.7절은 공통 명세 아래 12과제·8조건·3회로 생성한 288개 산출물의 평가를 설명한다. 요인 비교 결과는 4.4절에 제시한다.

C1~C4 비교는 6개의 소프트웨어 개발 과제에 대해 4가지 프롬프트 조건을 적용하고, 각 조건·과제 조합을 3회 반복하는 방식으로 수행하였다. 따라서 전체 코드 인스턴스는 다음과 같다.

$$6 \text{ tasks} \times 4 \text{ conditions} \times 3 \text{ runs} = 72 \text{ instances}$$

LLM 기반 코드 생성에는 Claude Code CLI의 비대화식 호출 방식인 claude -p를 사용하였다. 실제 코드 생성 모델은 claude-sonnet-5로 고정하였다. Temperature 및 출력 토큰 한도를 별도로 지정하지 않고 CLI 기본값을 사용하였다. 이에 따른 출력 변동성을 고려하여 각 과제·조건 조합을 3회 반복 생성하였으며, 반복 실행 결과는 조건별 기술통계에 포함하고, 통계 비교에서는 과제·조건별 세 반복을 요약하였다.

모델 원응답을 보존하고 추출한 코드에 동일한 평가 파이프라인을 적용하여 M1~M4 값을 산출하였다.

### 3.2 Experimental Task Design

과제는 실제 개발에서 요구되는 단순 함수부터 인증·파일 처리·권한 제어·결제 등 복합 기능까지 선정하였다(Table 2).

과제 복잡도는 요구되는 보안 요소 수, 외부 의존성, 상

www.kci.go.kr

Table 2. Software Development Tasks

| ID | Task | Complexity | Key Functions | Key Security Considerations |
|---|---|---|---|---|
| T1 | Password Validator | Level 1 | Password-strength validation function | Input validation, regular-expression safety |
| T2 | API Rate Limiter | Level 1 | Rate limiter implementation | IP-based rate limiting, boundary-value handling |
| T3 | JWT-Based Authentication API | Level 2 | User registration, login, and token validation | SQL injection, prevention of plaintext password storage, JWT validation |
| T4 | File Upload Handler | Level 2 | File reception, storage, and serving | Path traversal, MIME type validation, file-size limits |
| T5 | User-Authenticated Web Bulletin Board | Level 3 | User registration, login, and post CRUD operations | RBAC, XSS, SQL injection |
| T6 | Payment Processing REST API | Level 3 | Payment requests, validation, and transaction history management | Missing authentication, input tampering, idempotency |

태 관리 복잡도를 기준으로 Level 1부터 Level 3까지 구분하였다. Level 1은 단순 기능 중심의 과제이며, Level 2는 데이터베이스 또는 파일 입출력과 같은 외부 상태를 포함하는 과제이다. Level 3은 인증, 권한 제어, 이력 관리 등 복수의 컴포넌트가 결합된 복합 시스템 과제이다.

각 Level에는 두 과제만 포함되므로, 이 분류를 일반적인 난이도 척도로 보거나 난이도의 독립적 효과를 추정하지 않는다.

각 과제는 C3 및 C4 조건에서 Python 3.11과 Flask 기반 구현을 목표로 명시하였다. 반면 C1과 C2는 과제 수행에 필요한 최소 맥락을 제외하고, 세부 기술 스택, 보안 제약, 아키텍처 및 테스트 조건을 체계적으로 명시하지 않았다. 이는 바이브 코딩 환경에서 사용자가 모호하거나 단순한 자연어 요청만 제공하는 상황을 실험적으로 재현하기 위한 설계이다.

다만 T2와 T4의 C1 요청에는 Flask가 명시되어 있다. 동일한 스택과 인터페이스를 모든 조건에서 통제한 설계는 아니다.

## 3.3 Prompt Condition Design

### 3.3.1 C1 - Pure Vibe

C1은 기능 요건만 간략히 제시하는 비구조화 단일 프롬프트 조건이다. 기술 스택, 보안 요구사항, 출력 파일 구조, 테스트 조건은 명시하지 않으며, 일부 과제에는 Flask 등 최소 맥락이 포함된다. 예를 들어 T3 과제의 C1 프롬프트는 '로그인 기능이 있는 REST API를 만들어줘'와 같다. 이 조건은 바이브 코딩 사용자가 빠르게 결과물을 얻기 위해 최소한의 자연어 요청만 입력하는 상황을 재현한다.

### 3.3.2 C2 - Iterative Vibe

C2의 Turn 1은 C1과 같으며, Turn 2~5에서 입력 검증, 인증·인가, 보안·예외 처리 및 코드 구조화 요구를 순차적으로 추가하였다.

후속 지시는 실험 전에 과제별로 고정하고, 직전 응답에서 문제가 해결되었는지와 관계없이 같은 순서로 적용하였다. 따라서 C2는 실제 결함에 맞춰 지시를 바꾸는 적응형 디버깅이 아닌 고정된 반복 개선 조건이다.

Claude Code CLI의 비대화식 호출은 호출 간 대화 상태를 유지하지 않으므로, 매 턴 이전 사용자 요청과 모델 응답을 텍스트 형태로 누적하고 현재 턴의 후속 지시를 추가하여 새로운 프롬프트를 구성하였다. 최종 평가는 Turn 5 응답에서 추출한 코드를 대상으로 수행하였다.

### 3.3.3 C3 - Structured Prompt

C3는 단일 프롬프트 안에 역할, 기술 스택, 기능 요구사항, 보안 요구사항, 출력 형식을 명시하는 구조화 프롬프트 조건이다. 모든 C3 프롬프트는 다음과 같은 구조를 갖는다.

1. 역할 부여: OWASP Top 10을 숙지한 10년 경력의 시니어 백엔드 개발자
2. 기술 스택 명시: Python 3.11 및 Flask 등
3. 기능 요구사항: 엔드포인트, 입력·출력 형식, 핵심 동작
4. 보안 요구사항: CWE 또는 OWASP 관련 방어 조건
5. 출력 형식: 파일 구조, 타입 힌트, docstring, 단일 책임 원칙 등

이 조건은 프롬프트 엔지니어링에서 제안되는 역할 부여, 제약 조건 명시, 출력 형식 지정이 LLM 생성 코드의 품질과 갖는 관계를 비교하기 위한 조건이다.

### 3.3.4 C4 - Test-Driven Vibe

C4는 구조화된 요구사항과 사전 작성된 pytest 테스트 케이스를 함께 제공하는 조건이다. 이 조건에서는 LLM이 구현해야 할 기능과 함께 반드시 통과해야 하는 테스트 케이스를 프롬프트에 포함한다. 테스트 케이스는 정상 동작, 경계값 입력, 보안 공격 입력을 포함하도록 설계하였다.

www.kci.go.kr

Table 3. Definitions of Prompt Conditions C1-C4

| Condition | Name | Characteristics | Execution Method | Purpose |
|---|---|---|---|---|
| C1 | Pure Vibe | A short natural-language request containing only functional requirements | One call | Establish a baseline for unstructured vibe coding |
| C2 | Iterative Vibe | One C1-level request followed by four pre-defined follow-up instructions | Five sequential calls | Examine fixed iterative refinement |
| C3 | Structured Prompt | Includes a role, technology stack, security requirements, and output format | One call | Examine structured requirements |
| C4 | Test-Driven Vibe | Detailed requirements and pytest examples; specifications may differ from C3 | One call | Examine provided test specifications |

Table 4. Quality Metrics and Evaluation Tools

| Metric | Quality Attribute | Measurement | Evaluation Tool |
|---|---|---|---|
| M1 | Functional Correctness | Test pass rate, numbers of passed and total tests, and presence of syntax errors | pytest, pytest-json-report |
| M2 | Security | Number of warnings by research-defined category and weighted score | Bandit 1.8.0 |
| M3 | Complexity | Mean and maximum cyclomatic complexity and number of analyzed blocks | Radon 6.0.1 |
| M4 | Maintainability | Number of selected Pylint messages and distribution by category | Pylint 3.3.3 |

T3·T4·T6에서는 C3와 C4의 테스트 외 요구사항에도 차이가 있어, 모든 과제에서 C3에 테스트만 추가한 완전 중첩 설계는 아니다. 또한 제공 테스트와 평가 테스트가 상당 부분 중복되므로 C4의 높은 통과율을 독립 hidden test에 대한 일반화 성능으로 해석하지 않는다. 두 테스트 집합의 관계는 3.5.1절에 제시한다.

#### 3.3.5 Common Output Format Prefix

실제 코드 수집 과정에서는 C1-C4 모든 조건의 프롬프트 앞에 동일한 출력 형식 지시문을 추가하였다. 이는 LLM 응답에서 코드 추출을 안정적으로 수행하기 위한 실험 환경 정규화 조치이다.

공통 접두어는 다음과 같다.

- 다음 요청에 대해 완성된 코드만 출력해주세요.
- 마크다운 코드 블록(```python ... ```)으로 감싸서 텍스트로만 응답하세요.
- 파일 저장, 권한 요청, 설명 텍스트는 포함하지 마세요.

접두어는 응답 형식만 제한하며 기능·보안·파일 구조·테스트 요구를 추가하지 않는다. 조건 간 정보량·기술 스택·보안 명세·호출 횟수 차이는 유지된다.

### 3.4 Code Generation and Collection Procedure

Python 수집 스크립트가 과제 ID·조건·반복 번호에 따라 사전 정의된 프롬프트를 Claude Code CLI에 전달하고 응답의 Python 코드를 추출·저장하였다.

C1·C3·C4는 단일 호출, C2는 3.3.2절의 이력 누적 방식으로 5회 호출하여 최종 응답을 평가하였다.

C1~C4 조건에서 생성된 코드는 연구자가 수정하지 않았으며, 모델의 원본 응답과 함께 저장하였다. 코드 블록은 추출 규칙에 따라 파일별로 분리하거나 하나의 파일로 합쳐 저장하였다. 동일 인스턴스를 재수집할 때에는 기존 산출물을 삭제한 후 동일한 절차로 다시 생성하여 저장하였다.

### 3.5 Evaluation Metrics and Tools

C1~C4의 M1~M4는 동일한 테스트 하네스와 분석 도구로 산출하였다(Table 4). 모든 산출물은 Windows 11의 Python 3.12 환경에서 자동 평가하였다.

C3·C4가 요구한 Python 3.11은 생성 목표 버전이며, 실제 평가 버전과 구분한다. 동일한 평가기를 적용한 결과라도 프레임워크·엔드포인트·데이터베이스 인터페이스의 불일치를 포함할 수 있다.

www.kci.go.kr

Table 5. Test Composition and Overlap with C4-Provided Tests.

| Task | Test Composition | Evaluation Tests | C4-Provided Tests | Matching Names |
|---|---|---|---|---|
| T1 | Normal: 2; boundary-value: 6; security/exception: 4 | 12 | 10 | 10 |
| T2 | Normal: 2; boundary-value: 3; security/exception: 2 | 7 | 6 | 6 |
| T3 | User registration: 3; login: 5; authentication protection: 3; security attributes: 1 | 12 | 12 | 12 |
| T4 | Normal: 3; boundary-value: 2; security: 4 | 9 | 8 | 8 |
| T5 | Authentication: 4; post CRUD operations: 5; security/authorization: 3 | 12 | 11 | 11 |
| T6 | Normal: 3; idempotency: 1; input validation: 4; security/authorization: 2 | 10 | 8 | 8 |
| Total | Six tasks | 62 | 55 | 55 |

#### 3.5.1 Functional Correctness (M1)

기능 정확성은 생성 코드가 과제별로 사전 정의된 테스트를 통과한 비율로 측정하였다. 모든 프롬프트 조건에 동일한 pytest 테스트셋을 적용하였으며, 테스트 실행 결과는 pytest-json-report를 통해 구조화된 형식으로 수집하였다. 데이터베이스 기능을 포함하는 과제에서는 기능 정확성 평가를 SQLite 데이터베이스 환경에서 수행하였다.

기능 정확성은 아래 식과 같이 계산하였다.

$$M_{1,i} = \begin{cases} \frac{P_i}{T_i} \times 100 & (T_i > 0) \\ 0 & (T_i = 0) \end{cases}$$

여기서 $P_i$는 인스턴스 $i$가 통과한 수이며, $T_i$는 해당 과제의 전체 테스트 수이다. 분석에서는 테스트별 결과를 모든 과제에 걸쳐 직접 합산하지 않고, 각 인스턴스에서 통과율을 계산한 뒤 과제·조건별 반복값을 요약하였다. 이에 따라 테스트 수가 많은 과제에 더 큰 가중치가 부여되지 않도록 하였다. C1~C4에 적용한 평가기는 skipped와 error를 전체 수에 포함하고, 전체 수가 0이면 통과율을 0으로 기록하므로 이 값만으로 기능 결함을 확정하지 않는다.

과제별 테스트 수는 기능·보안 요구사항의 검증 범위에 따라 정하였다. 동일한 수를 강제할 때 생기는 중복 검사나 필수 요구사항 누락을 피하고자 요구사항 커버리지를 우선하였다(Table 5).

저장 로그에서 테스트 수집 수와 실제 실행 수를 구분하였다. 실제 실행은 pytest의 call 단계가 존재하고 skipped가 아닌 경우로 정의하였다. 전체 스킵, 수집·구문 오류, 인터페이스·의존성 불일치, 실행 중 예외 및 단언 실패를 별도로 집계하였다. 로그에서 생성 코드와 평가 코드 중 예외의 원인을 확정할 수 없으면 구분 불가로, 실행 0건의 조건부 통과율은 결측값으로 처리하였다.

#### 3.5.2 Security Warnings (M2)

보안 경고는 Python 정적 분석 도구 Bandit을 사용하여 측정하였다. Bandit이 보고한 CWE 식별자를 본 연구의 심각도 분류 기준에 매핑하여 Critical, High, Medium 및 Low로 분류하였다. 사전 정의된 CWE 목록에 포함되지 않은 이슈는 Bandit이 제공하는 기본 심각도를 적용하였다.

주요 분석에는 Critical, High 및 Medium 보안 경고의 개수와 가중 점수를 사용하였다.

가중 점수는 실제 평가 스크립트에 구현된 아래 식에 따라 계산하였다.

$$M2_{\ weighted} = 3N_{Critical} + 2N_{High} + N_{Medium}$$

여기서 N은 각 범주에 해당하는 경고 수이다. Low 이슈는 세부 결과에 기록하되 가중 점수에는 포함하지 않았다. 가중 점수가 낮을수록 이 집계 기준에 따른 경고 수준이 낮음을 의미하며, 실제 악용 가능성이나 수동 검증된 취약점 수를 뜻하지 않는다. 평가 스크립트는 confidence 임계값으로 필터링하지 않았고 개별 confidence 및 수동 오탐 판정 기록을 보존하지 않았다.

보조 코드를 제외한 저장 코드를 Bandit 1.8.0으로 재분석하여 severity와 confidence를 확인하였다. 검출된 43개 경고는 모두 MEDIUM confidence로, MEDIUM/HIGH 기준을 적용해도 경고 수는 같았다. 이 신뢰도 검토는 CWE 가중 점수와 별도로 수행했으며 수동 오탐 검증은 포함하지 않았다.

www.kci.go.kr

#### 3.5.3 Code Complexity (M3)

코드 복잡도는 Radon을 사용하여 사이클로매틱 복잡도를 측정하였다. 각 인스턴스에 대해 저장된 블록별 복잡도의 평균 값, 최대값 및 블록 수를 기록하였다.

저장 블록에는 함수뿐 아니라 클래스도 포함되므로 순수 함수·메서드 평균과 구분한다. 집계된 평균 복잡도에 적용한 등급은 개별 블록의 등급과 같지 않아 주요 비교에 사용하지 않았다.

Radon이 구문 오류로 분석하지 못한 파일은 제외하고 정상 파싱된 파일만 집계하였다. M3는 M1의 구문 오류 여부 및 분석 블록 수와 함께 확인하였다.

#### 3.5.4 Code Smells (M4)

M4는 Pylint 3.3.3의 정적 분석 메시지 중 오류성 메시지(E 계열)를 제외한 경고·규약·리팩토링 메시지로 측정하였다.

집계 대상 메시지 ID는 C0301·C0302, R0801, R0913, W0611·W0612·W0613, W0109·R2004, C0103·C0104, R1702, C0114·C0115·C0116 및 나머지 R 계열이다. 범주 이름만으로 실제 메시지 의미를 오해하지 않도록 ID를 기준으로 정의한다.

각 인스턴스에 대해 전체 코드 스멜 수와 카테고리별 발생 수를 기록하였으며, 본 분석에서는 전체 코드 스멜 수를 유지보수성의 제한된 대리 지표로 사용하였다.

C3·C4의 docstring 및 명명 요구가 평가 항목과 중복되므로, 문서화·명명 메시지를 제외한 보조 집계도 함께 확인하였다.

#### 3.5.5 Sensitivity Analysis of Metric Composition

평가 과정에서 추가된 보조 파일도 C1~C4의 정적 분석에 포함되어 있어, 저장된 경고·블록·메시지 목록에서 해당 파일의 항목을 제외하여 M2~M4를 다시 집계하였다. M4에는 문서화·명명 항목을 추가로 제외한 축소 지표를 사용하였다. 이 절차는 저장 목록의 사후 필터링이며, 파일 간 중복 탐지나 블록 처리 등 평가 도구의 전체 동작을 다시 실행한 결과와 같지 않다.

규모 차이를 확인하기 위해 주석·빈 줄·docstring과 보조 코드를 제외한 소스 행 수(SLOC)를 집계하고, 생성 코드의 Bandit 전체 경고와 축소 M4를 각각 1,000 SLOC당 개수로 환산하였다. 이는 보조 기술통계이며, 짧고 불완전한 코드의 기능 누락이나 분석 범위 차이까지 보정하지는 않는다.

### 3.6 Statistical Analysis Methods

M1~M4의 평균, 표준편차, 중앙값 및 사분위범위를 산출하였다. 추론 분석에서는 과제·조건별 세 반복의 평균으로 6×4 행렬을 구성하고, 과제를 독립 분석 단위로 삼았다.

RQ1에서는 과제를 블록으로 둔 Friedman 통계량 Q에 동률 보정을 적용하였다. 과제가 6개인 소표본을 고려하여 과제 내 조건 라벨을 199,999회 무작위 순열하고, 관찰값 이상의 순열 통계량 수를 b, 순열 수를 B라 할 때 p=(b+1)/(B+1)로 계산하였다. 네 지표의 전체 검정에는 Holm 보정을 적용하였으며 유의수준은 0.05로 설정하였다. 순열 seed는 M1~M4 순서로 20260911~20260914이다. 과제별 순위 일관성은 Kendall W=Q/[6×(4-1)]로 제시하였다.

쌍별 비교에는 과제별 차이의 영값을 제외하고 절대차에 동률 평균 순위를 부여한 양측 부호순위 검정을 적용하였다. 가능한 부호배열을 전수 열거하여 정확 p값을 구하고, 지표별 여섯 조건쌍에 Holm 보정을 적용하였다. 대응 순위 효과크기 $r_{rb} = \frac{W^+ - W^-}{W^+ + W^-}$와 비영 차이 과제 수를 함께 보고하였다. 차이는 앞 조건에서 뒤 조건을 뺀 값이며, M1은 높을수록, M2~M4는 낮을수록 해당 관찰 지표에서 양호한 방향이다.

RQ2에서는 Level별 두 과제와 과제별 세 반복, 즉 조건별 6개 산출물의 중앙값·사분위범위·분포를 탐색적 기술통계로 비교하였다.

RQ3에서는 RQ1의 C2-C3, C2-C4 및 C3-C4 쌍별 비교를 활용하고, 테스트 명세 노출과 3.5.5절의 지표 구성 변경에 따른 해석을 함께 비교하였다. 과제별 세 반복을 평균 대신 중앙값으로 요약한 분석도 민감도 확인에 사용하였다. 각 민감도 분석의 네 전체 검정에는 별도로 Holm 보정을 적용하였다.

순열 검정은 과제 내 조건의 교환가능성과 과제 간 독립성을 가정하는 사후 탐색 분석이다. 독립 과제가 6개여서 쌍별 검정의 해상도와 검정력에 제약이 있다.

### 3.7 Controlled Factorial Design

모든 조건에 Python 3.12, Flask 3.1.0, 동일한 앱 생성 인터페이스 및 기능·입력 타입·오류 응답·경계값·접근 권한 명세를 제공하였다. 과제는 비밀번호 검증, 요청 제한, 인증, 파일 보관, 게시판, 결제, 재고, 예약, 단축 URL, 서명 이벤트, 목록 조회, 쿠폰의 12개 편의 표본이며, 검증된 난이도 수준을 부여하지 않았다.

www.kci.go.kr

프롬프트 형식(F), 추가 보안 구현 지침(S), 공개 예시 테스트(T)를 각각 2수준으로 조작하였다. F0은 연속 문단, F1은 제목·항목 구분을 사용하며 단어 순서는 같다. S0과 S1의 기능·접근 권한 의무는 동일하고 S1에만 안전한 구현 지침을 추가하였다. T1에는 기본 동작의 공개 예시 2개를 제공하고 T0에는 제공하지 않았다. 8개 조합을 과제별로 3회 독립 생성하여 288개를 분석하였다.

Claude Code 2.1.292에서 claude-sonnet-5와 high effort를 고정하고, 도구·프로젝트 문서·사용자 설정·대화 이력을 차단한 단일 요청으로 생성하였다. 실제 응답의 모델 식별자를 확인했고 온도·출력 토큰 한도는 CLI 기본값을 사용하였다. 수집 순서는 결과 확인 전에 무작위화했으며, 반복 대화는 비교 요인에 포함하지 않았다.

과제별 공개 예시 2개와 미제공 시나리오 8개를 구성하였다. 미제공 평가는 공통 명세의 경계값·잘못된 타입·상태 전이·권한·중복 요청을 공개 예시와 다른 검증 목적·입력·기대 결과로 확인하였다. 시나리오 내 모든 검증이 성공해야 통과로 판정하였다. 입력과 평가 사례는 생성 전에 해시로 고정하고 수집 종료 후 평가하였다. 평가 구현 오류는 결과 열람 전에 정정·기록했으며 입력·사례·표본 수·가설은 유지하였다. 결과에 따른 코드 수정·재생성은 하지 않았다.

생성 완료 후 평가 전 명세 대조에서 목록 조회 과제의 공통 ID 타입과 정수 ID 규칙의 충돌을 확인하였다. 생성 코드·평가 결과를 열람하기 전에 모든 조건의 정수 ID 보존을 명확히 하고 해당 과제의 24개를 전부 재수집하였다. 수정 전 24개는 보존하되 분석에서 제외했으며, 다른 11과제·공개 및 미제공 평가·모델·분석 기준은 유지하였다.

정답 구현은 공개 24개·미제공 96개를 모두 통과했고, 선택한 결함 변형 12개는 공개 예시를 통과하면서 미제공 평가에서 검출되었다. 이 평가 도구 검증 자료는 모델 성능 집계에서 제외하였다. 생성 코드는 동일한 네트워크 차단 컨테이너에서 실행하였다. 주 지표는 예정 시나리오 중 통과 비율이다. 수집 수는 등록 시나리오, 실행 수는 실제 요청 실행에 도달한 시나리오 수로 정의하고 구문·불러오기·인터페이스·설정 오류, 실행 중 예외, 단언 실패 및 시간 초과를 별도 집계하였다. 실행된 사례 중 통과율도 계산하되 실행 0건은 결측 처리하였다.

보안 경고 점수는 Bandit의 MEDIUM/HIGH confidence 경고에 도구 고유 severity LOW/MEDIUM/HIGH별 1/2/3을 부여해 합산하였다. 이는 3.5.2절의 CWE 가중치와 구분한다. 함수·메서드·중첩 함수의 평균 순환 복잡도와 문서화·명명을 제외한 Pylint 메시지 수는 기술통계로 보고하였다. 후자는 3.5.4절의 선택 목록과 집계 범위가 다르다. 모든 정적 지표는 생성 코드만 대상으로 하며 분석 실패는 결측 처리하였다.

독립 분석 단위는 12과제이다. 과제·조건별 세 반복을 평균한 뒤 다른 두 요인의 수준을 동일 가중 평균하여 각 요인의 1수준과 0수준 차이를 구하였다. 과제별 차이의 부호 조합 4,096개를 전수 열거한 양측 정확 검정을 적용하였다. 귀무가설하 과제 효과의 부호 대칭과 과제 간 독립성을 가정하며 유의수준은 0.05이다. 미제공 통과율의 세 주효과와 보안 경고 점수의 세 주효과는 각각 별도 검정군으로 Holm 보정하였다. 복잡도·Pylint 메시지·개별 조건 순위는 기술통계로 제시하였다.

외부 지시는 실행마다 한 번 전달했으나 CLI 내부 턴 수가 2로 기록된 응답이 두 개 있었다. 저장 메타데이터에서 원인을 확정할 수 없어 응답을 보존하고, 결과 열람 전 정한 기준에 따라 해당 두 과제 블록을 제외한 보조 분석을 수행하였다. 주 분석은 12과제를 유지하였다.

## IV. Experimental Results

본 장에서는 C1~C4 조건에 따른 생성 코드의 품질 차이와 과제 복잡도별 특성을 살펴본다. 이어서 공통 명세 아래 프롬프트 형식, 추가 보안 구현 지침 및 예시 테스트 제공에 따른 차이를 분석한다.

### 4.1 RQ1: Quality Comparison Across Prompt Conditions

C1~C4 조건의 저장 로그에서 조건별 테스트 수집 수는 C1~C4 순서로 186, 186, 174, 186개였고 실제 실행 수는 117, 117, 117, 172개였다. 조건별 18회 중 실행 0건은 각각 6, 6, 6, 1회였다. 전체 스킵은 3, 4, 2, 0회, 전체 오류는 3, 2, 3, 1회였으며 C3에는 별도의 구문 오류 1회가 있었다. 따라서 아래 통과율은 실제 기능 단언뿐 아니라 평가 하네스와의 호환성을 포함한 종합 결과이다.

C1~C4 조건의 테스트 통과율과 보안 경고 점수는 과제별 반복 평균을 이용한 Friedman 순열 검정에서 전체 조건 차이가 유의하였다. Holm 보정 p값은 각각 0.011930과 0.001395였으며 Kendall W는 0.6049와 0.7593이었다. 보조 코드를 포함한 복잡도와 코드 스멜 집계도 전체 차이가 유의하였으나(Holm 보정 p=0.029590, 0.000780; Kendall W=0.4778, 0.8164), 4.3절의 민감도 분석에서는 유의성이 유지되지 않았다.

테스트 통과율의 중앙값은 C1과 C2에서 모두 0%, C3에

서 40.715%, C4에서 100%로 나타났으며, C4의 평균은 85%였다. 보안 경고 가중 점수의 중앙값은 C1과 C2에서 각각 2.0점, C3와 C4에서 모두 0점이었다. 평균 복잡도의 중앙값은 C1 3.645, C2 3.760, C3 3.100, C4 3.395였으며, 코드 스멜 수는 각각 11.0개, 12.5개, 6.0개, 6.5개였다.

Fig. 1은 과제별 반복 평균을 연결하여 보여준다. M1은 위 통과율과 동일한 지표이며, M2~M4는 평가 보조 코드를 제외한 집계이다. M4는 문서화·명명 항목도 제외하였다. 지표 구성에 따른 비교는 4.3절에 제시한다.

정확 부호순위 검정에서 지표별 여섯 쌍을 보정한 결과, 유의한 조건쌍은 확인되지 않았다(Table 6). 보정 p값의 최솟값은 0.1875였다.

### 4.2 RQ2: Descriptive Comparison by Task Complexity

과제 복잡도에 따른 조건별 결과를 Level 1~3으로 구분하여 비교하였다. 각 Level은 두 과제에서 얻은 조건별 6개 산출물로 구성되므로 중앙값과 분포의 변화 양상을 탐색하였다.

테스트 통과율에서 C4는 Level 1~3의 중앙값이 각각 100%, 87.5%, 90%로 가장 높았다. Level 3에서는 C1과 C2가 모두 0%, C3가 5%로 나타나, C4가 다른 조건보다 높은 통과율을 보였다. 보안 경고 가중 점수는 Level 2에서 C1과 C2의 중앙값이 모두 2점인 반면 C3와 C4는 모두 0점이었다. Level 3에서도 C3와 C4는 0점으로 낮은 경고 점수를 유지하였다.

Fig. 2는 과제별 테스트 통과율과 축소 코드 스멜 수를 세 반복의 평균으로 나타낸다. Level 3의 T5와 T6에서 C4의 평균 통과율은 각각 58.3%와 93.3%였으며, C1과 C2는 두 과제 모두 0%였다. 반면 축소 코드 스멜 수에서는 과제별 조건 순위가 달랐다. 예를 들어 T5에서는 C2가, T6에서는 C3가 가장 낮은 값을 보였다.

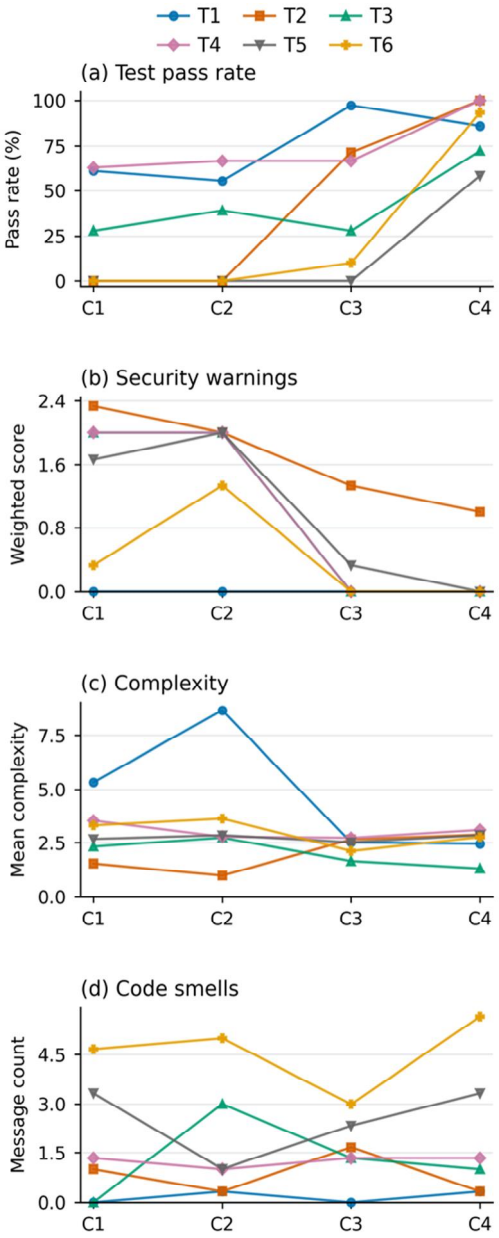


Fig. 1. Code Quality Across Prompt Conditions

Table 6. Pairwise Comparisons of Original Metrics

| Comparison | M1: p/r/n | M2: p/r/n | M3: p/r/n | M4: p/r/n |
|---|---|---|---|---|
| C1-C2 | 0.8750 / -0.333 / 3 | 1.0000 / -0.500 / 3 | 0.4688 / -0.524 / 6 | 1.0000 / -0.143 / 6 |
| C1-C3 | 0.3750 / -0.867 / 5 | 0.3750 / 1.000 / 5 | 0.3125 / 0.905 / 6 | 0.1875 / 1.000 / 6 |
| C1-C4 | 0.1875 / -1.000 / 6 | 0.3750 / 1.000 / 5 | 0.4688 / 0.619 / 6 | 0.1875 / 1.000 / 6 |
| C2-C3 | 0.8750 / -0.467 / 5 | 0.3750 / 1.000 / 5 | 0.1875 / 1.000 / 6 | 0.1875 / 1.000 / 6 |
| C2-C4 | 0.1875 / -1.000 / 6 | 0.3750 / 1.000 / 5 | 0.4688 / 0.714 / 6 | 0.1875 / 1.000 / 6 |
| C3-C4 | 0.2500 / -0.905 / 6 | 1.0000 / 1.000 / 2 | 0.3125 / -0.905 / 6 | 1.0000 / -0.067 / 5 |

* Each cell reports Holm-adjusted p / paired rank-biserial effect / number of nonzero task differences. Differences are first condition minus second condition; higher M1 and lower M2-M4 indicate favorable directions.

www.kci.go.kr

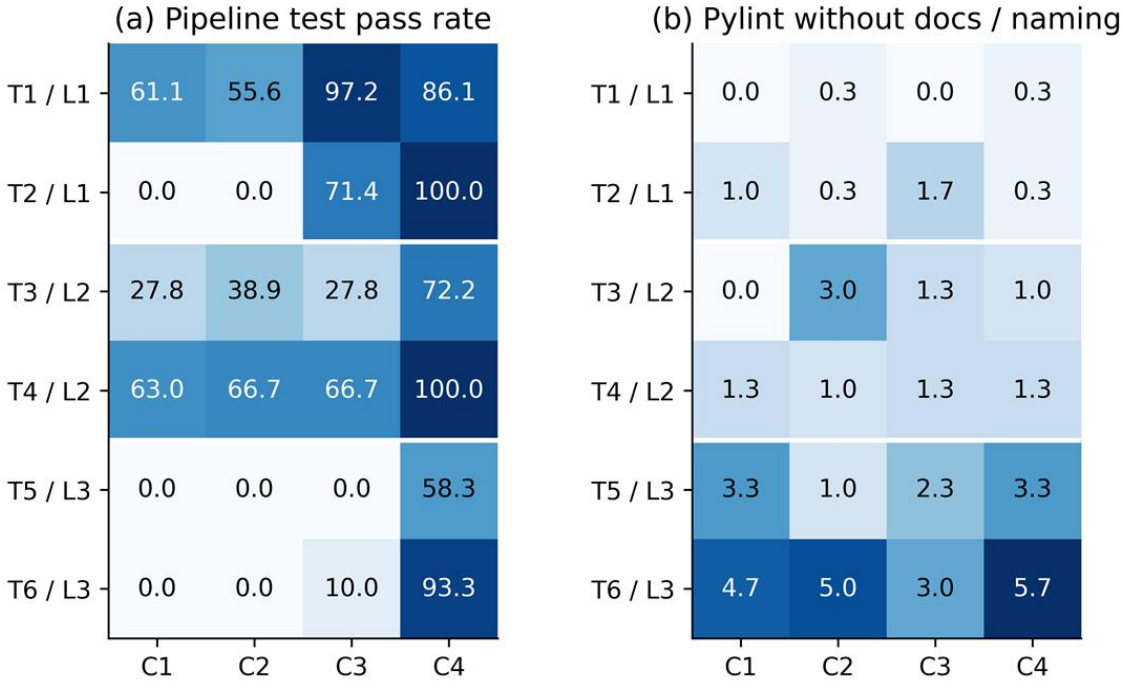


Fig. 2. Code Quality by Task

### 4.3 RQ3: Comparison of Prompting Strategies

RQ3는 C2의 고정된 후속 지시, C3의 구조화 명세, C4의 테스트 명세 제공 조건을 RQ1의 쌍별 비교와 민감도 분석으로 살펴보았다.

C4는 과제별 반복 평균에서도 여섯 과제 모두 C1과 C2보다 높은 통과율을 보였다. C3와 C4의 보안 경고 점수 중앙값은 모두 0점으로 C2의 2점보다 낮았다. C4에는 평가와 상당 부분 중복되는 테스트가 제공되었다.

3.5.5절의 민감도 분석에서도 M1과 M2의 전체 조건 차이는 유의하게 유지되었다(Holm 보정 p=0.017895, 0.001860). 반면 평가 보조 코드를 제외한 M3와 문서화·명명을 추가로 제외한 축소 M4에서는 유의한 전체 차이가 확인되지 않았다(Holm 보정 p=0.635160, 0.837300). 축소 M4의 중앙값은 C1 0.5개, C2 1.0개, C3 1.5개, C4 1.0개로, 축소 전 코드 스멜 수에서 나타난 조건별 순위와 달랐다.

코드 규모를 고려한 보조 분석에서 생성 코드의 1,000 SLOC당 Bandit 전체 경고 수 중앙값은 C1 14.146건, C2 14.815건, C3와 C4 모두 0건이었다. 축소 M4의 동일 규모당 중앙값은 각각 2.463개, 12.803개, 8.501개, 10.153개였다. 과제별 반복을 평균 대신 중앙값으로 요약해도 M1·M2·M4의 전체 차이는 유의하였으나, 보조 코드를 포함한 M3의 전체 순열 p값은 0.109110으로 변하였다.

C1~C4의 쌍별 비교에서는 유의한 차이가 확인되지 않았으며(Table 6), 공통 명세 아래 개별 요인의 비교 결과는 4.4절에 제시한다.

### 4.4 Factorial Comparison Results

공통 명세 아래 생성한 288개 산출물에서 계획된 미제공 시나리오 2,304개가 모두 실행되었고 2,266개가 통과하였다. 평균 통과율은 98.35%였으며, 283개 산출물이 과제별 8개 시나리오를 모두 통과했다. 실행 불능, 인터페이스 불일치, 스킵 및 시간 초과는 없었다. 실패 38개는 다섯 산출물의 응답 검증 단계에서 발생하였다.

조건별 평균 통과율은 F0S0T0에서 94.79%, F0S1T0에서 92.01%였고 나머지 여섯 조건에서는 100%였다. 미통과 산출물은 F0S0T0에서 2개, F0S1T0에서 3개로 모두 F0·T0 조합에 속했다. 실패 사례는 요청 제한, 서명 이벤트 및 목록 조회의 일부 구현에서 사용자 식별 헤더를 불필요하게 요구하여 HTTP 401을 반환한 경우였다.

Table 7은 과제별 세 반복과 다른 두 요인의 수준을 동일 가중 평균하여 구한 각 요인의 1수준과 0수준 간 통과율 차이(%p)를 나타낸다. 형식 F와 예시 T의 차이는 각각 +3.30%p, 추가 보안 구현 지침 S는 -0.69%p였으며, 세 주효과 모두 Holm 보정 후 유의하지 않았다. 모든 실패가 F0·T0 조합에 집중되어 F와 T의 0수준 평균이 동일하게

www.kci.go.kr

낮아졌으며, 0이 아닌 차이는 12과제 중 세 과제에서 나타났다. 두 수치는 동일한 실패 양상을 요인별로 요약한 주변 비교값으로, 각 요인의 독립적인 성능 개선을 입증한 것으로 해석하지 않는다.

보안 경고는 HIGH confidence의 LOW severity B311 한 개였고, F·S·T의 보조 검정은 모두 Holm 보정 p=1.00이었다. 산출물별 평균 함수 복잡도는 2.6~11.0, 문서화·명명을 제외한 Pylint 메시지 수는 0~9개였다.

CLI 내부 턴 수 예외가 포함된 두 과제를 제외한 10과제 보조 분석에서 F와 T의 차이는 각각 +3.125%p, S는 0%p였다. 세 Holm 보정 p값은 모두 1.00이었다.

Table 7. Marginal Effects on Hidden Test Pass Rates

| Factor | Difference (pp) | Exact p | Holm p |
|---|---|---|---|
| F | +3.299 | 0.25 | 0.75 |
| S | -0.694 | 1.00 | 1.00 |
| T | +3.299 | 0.25 | 0.75 |

## V. Conclusions

본 연구는 프롬프트 조건에 따른 바이브 코딩 생성물의 품질 차이를 탐색하였다. 6과제에 C1~C4 조건을 적용한 72개 산출물을 비교하고, 공통 기능·기술 스택·인터페이스 명세 아래 12과제·8조건의 288개 산출물을 별도로 평가하였다. 공통 명세에서는 프롬프트 형식, 추가 보안 구현 지침 및 공개 예시 테스트의 세 요인을 독립적으로 조작하였다.

C1~C4의 복합 조건 비교에서 테스트 통과율과 보안 경고 점수의 전체 조건 차이는 민감도 분석에서도 유의하게 유지되었다. C4의 통과율 중앙값은 100%, 평균은 85%였지만 제공 테스트와 평가가 상당 부분 중복되었다. 실행 로그 분석에서는 C1·C2·C3의 각 18회 중 6회, C4의 18회 중 1회에서 테스트가 실행되지 않았다. 따라서 C1~C4의 통과율 격차에는 평가 호환성이 포함되며, 특정 전략의 독립 효과로 해석하지 않는다. 보조 코드를 포함한 복잡도와 코드 스멜의 전체 유의성도 민감도 분석에서 유지되지 않아 안정적인 개선 근거로 제시하지 않았다.

공통 명세 아래의 요인 비교에서는 미제공 시나리오 2,304개가 모두 실행되었고 평균 통과율은 98.35%였다. 형식 구조화와 공개 예시 제공의 주변 평균 차이는 각각 +3.30%p였으나 Holm 보정 p=0.75로 유의하지 않았다. 추가 보안 구현 지침의 차이는 -0.69%p, 보정 p=1.00이었다. 실패는 세 과제의 다섯 산출물에서 조건부 사용자 식별 요구를 과도하게 적용한 사례에 집중되었다. 보안 경고는 한 개뿐이어서 지침의 효과를 판별하기 어려웠다.

이 결과는 복합 프롬프트 조건에서 관찰된 품질 차이와 공통 명세 아래 개별 요인의 추가 효과를 구분하여 해석할 필요성을 보여준다. 288개 중 283개가 모든 미제공 시나리오를 통과하는 천장효과로 인해, 이 평가 환경에서는 추가적인 프롬프트 효과를 충분히 구별하는 데 제약이 있었다. 보안 경고도 희소하여 비유의성을 효과 부재나 동등성으로 해석하지 않는다. 본 연구의 기여는 복합 조건의 관찰 결과와 공통 명세 아래의 요인 비교 결과를 구분하고, 테스트 실행 상태·미제공 시나리오 통과율·정적 분석 지표를 함께 분석한 데 있다.

공통 명세와 미제공 평가를 적용한 요인 비교는 단일 모델과 12개 소규모 Flask 과제의 편의 표본에 한정된다. 공개 예시와 미제공 평가는 같은 명세를 공유하며, 과제당 8개 시나리오가 모든 결함을 포괄하지 않는다. F는 같은 내용의 배열 형식, S는 추가 보안 구현 지침, T는 두 공개 예시의 제공에 한정되므로 광범위한 프롬프트 전략이나 반복 대화 전체로 확대하지 않는다. C1~C4의 비교 결과는 조건 간 정보 차이와 테스트 중복을 고려하여 복합 조건의 관찰로 해석한다. 또한 보안 경고와 정적 분석 메시지는 수동 검증된 취약점이나 유지보수성 자체를 뜻하지 않는다. 향후 다양한 모델·언어·과제와 동적 보안 평가, 개발자 검토 및 적응형 반복 조건을 통해 적용 범위를 확인할 필요가 있다.

## REFERENCES

[1] A. Karpathy, "Vibe Coding," X, February 2025. [Online]. [Accessed: July 2, 2026]. Available: https://x.com/karpathy/status/1886192184808149383.

[2] M. Yoon and S. Park, "Developing Educational Web Applications through Generative AI-Based Vibe Coding: Applying Design-Based Research Methodology," The Journal of Korean Association of Computer Education, Vol. 29, No. 5, pp. 57-71, May 2026. DOI: 10.32431/kace.2026.29.5.006.

[3] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), pp. 754-768, San Francisco, CA, USA, May 2022. DOI: 10.1109/SP46214.2022.9833571.

[4] G. Sandoval, H. Pearce, T. Nys, R. Karri, S. Garg, and B. Dolan-Gavitt, "Lost at C: A User Study on the Security

www.kci.go.kr

Implications of Large Language Model Code Assistants," Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), pp. 2205-2222, Anaheim, CA, USA, August 2023. https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval.

[5] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv, arXiv:2107.03374, July 2021. https://arxiv.org/abs/2107.03374. DOI: 10.48550/arXiv.2107.03374.

[6] E. Ko and P. Kang, "Evaluating Coding Proficiency of Large Language Models: An Investigation Through Machine Learning Problems," IEEE Access, Vol. 13, pp. 52925-52938, March 2025. DOI: 10.1109/ACCESS.2025.3553870.

[7] J. S. Kim, O. Wang, J. H. Lee, H. G. Cho, and W. S. Lee, "Ensuring the Syntactic Correctness of Code Generated by LLMs," Proceedings of Korea Software Congress 2024 (KSC 2024), pp. 1181-1183, Yeosu, Republic of Korea, December 2024. https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE12042207.

[8] C. S. Xia, Y. Wei, and L. Zhang, "Automated Program Repair in the Era of Large Pre-Trained Language Models," Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 1482-1494, Melbourne, Australia, May 2023. DOI: 10.1109/ICSE48619.2023.00129.

[9] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," Proceedings of the 30th Conference on Pattern Languages of Programs (PLoP 2023), Article 5, pp. 1-31, Allerton Park, Monticello, IL, USA, October 2023. DOI: 10.64346/PLoP2023p05.

[10] S. Yeo, H. Lim, H. Jun, T. Kim, S. Kim, and Y. Ma, "Method of Combining Prompts for Code Generation Based on LLM," Proceedings of Korea Software Congress 2023 (KSC 2023), pp. 633-635, Busan, Republic of Korea, December 2023. https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE11705189.

[11] S. Choi, D. Lee, J. Kim, Y. Jang, and H. Kim, "Designing LLM-based Code Reviewing Learning Environment for Programming Education," The Journal of Korean Association of Computer Education, Vol. 26, No. 5, pp. 1-11, September 2023. DOI: 10.32431/kace.2023.26.5.001.

[12] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," Advances in Neural Information Processing Systems, Vol. 35, pp. 24824-24837, New Orleans, LA, USA, November-December 2022. DOI: 10.52202/068431-1800.

[13] C. Tony, M. Mutas, N. E. Díaz Ferreyra, and R. Scandariato, "LLMSecEval: A Dataset of Natural Language Prompts for Security Evaluations," Proceedings of the 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 588-592, Melbourne, Australia, May 2023. DOI: 10.1109/MSR59073.2023.00084.

[14] T. J. McCabe, "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, pp. 308-320, December 1976. DOI: 10.1109/TSE.1976.233837.

[15] M. Fowler, "Refactoring: Improving the Design of Existing Code," Addison-Wesley, pp. 75-88, 1999. ISBN: 978-0-201-48567-7.

[16] M. Yu, J. Choi, and Y. Peak, "A Survey on Secure Code Generation with Large Language Models," Proceedings of the Annual Conference of KIPS (ACK 2025), Vol. 32, No. 2, pp. 164-166, Seoul, Republic of Korea, November 2025. DOI: 10.3745/PKIPS.y2025m11a.164.

[17] OWASP Foundation, "OWASP Top 10: 2021," OWASP, https://top10.owasp.org/2021/.

[18] MITRE, "Common Weakness Enumeration," CWE, https://cwe.mitre.org/.

[19] PyCQA, "Bandit Documentation," Read the Docs, https://bandit.readthedocs.io/en/latest/.

## Authors



Chul-Hee Lee received the B.S. degree in information computer science from Konyang University, Korea, in 2015, and the M.S. degree in information security from Sejong Cyber University, Seoul, Korea, in 2026.

He is currently a visiting professor with the Department of Cloud Security, Korea Polytechnics, Daejeon, Korea. His research interests include data and 3D visualization, cloud security, and prompt engineering for vibe coding.



Sang-Yong Choi received his B.S. degree in Mathematics and M.S. degree in Computer Science, both from Hannam University in 2000 and 2003, and Ph.D. degree in Interdisciplinary of Information Security from Chonnam National University in 2014. Dr. Choi is an assistant professor at Dept. of Cloud Security in Korea Polytechnics, Daejeon, Korea. His research interests are in security education, web security, network security and cloud computing security.

www.kci.go.kr