Journal of KIIT. Vol. 24, No. 5, pp. 75-92, May 31, 2026. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2026.24.5.75

LLM 기반 코드 생성에서 구조화된 비즈니스 로직 표현의 효과 분석: Atomic Logic Sheet 접근법

김만수*[1], 최무성*[2], 왕은영*[3], 정성택**

Effect Analysis of Structured Business Logic Representation in LLM-based Code Generation: The Atomic Logic Sheet Approach

Mansu Kim*[1], Museong Choi*[2], Eunyoung Wang*[3], and Sungtaek Chung**

요 약

본 연구는 LLM 기반 코드 생성에서 설계 정보의 표현 방식이 코드 품질에 미치는 영향을 분석한다. 기존 자연어 중심 프롬프트는 비즈니스 로직 준수 저하, 규칙 충돌 미탐지, 스키마 할루시네이션 등의 한계를 가진다. 이에 비즈니스 로직을 마크다운 기반 구조로 표현한 ALS(Atomic Logic Sheet)를 제안하고, Claude Code CLI 기반 에이전트 환경에서 WMS 도메인 3그룹 비교 실험(A: 요구사항, B: 자연어 설계문서, C: ALS)을 수행하였다. 실험 결과, 로직 준수율은 A=90.7%, B=94.7%, C=96.1%로 설계 문서의 효과가 확인되었고, 충돌 탐지율은 A=0.0%, B=17.5%, C=100%로 ALS의 구조화 효과가 두드러졌다. 또한 스키마 일탈은 A=44.1, B=1.0, C=3.5로 나타나, ALS가 코드 생성의 추종성(Sycophancy)과 안정성 향상에 유의미하게 기여함을 확인하였다.

Abstract

This study analyzes the effect of design information representation on code quality in LLM-based code generation. Conventional natural language prompting approaches have limitations, including reduced compliance with business logic, failure to detect rule conflicts, and schema hallucination. To address this issue, we propose the Atomic Logic Sheet (ALS), a markdown-based structured representation of business logic, and conduct a three-group comparative experiment in a WMS domain using a Claude Code CLI-based agent environment (A: requirements only, B: natural language design document, C: ALS). The results show that Logic Compliance Rate (LCR) improved from A=90.7% to B=94.7% and C=96.1%, confirming the effect of design documentation. Conflict Detection Rate (CDR) was A=0.0%, B=17.5%, and C=100%, demonstrating the strong impact of ALS structuring. In addition, Schema Deviation Count (SDC) was A=44.1, B=1.0, and C=3.5, indicating that ALS contributes significantly to improving consistency and stability in code generation.

Keywords

large language model, atomic logic sheet, prompt engineering, AI sycophancy, schema hallucination

* 한국공학대학교 컴퓨터공학과
- ORCID[1]: https://orcid.org/0009-0001-1322-2679
- ORCID[2]: https://orcid.org/0009-0004-6166-3614
- ORCID[3]: https://orcid.org/0000-0002-3286-6496
** 한국공학대학교 컴퓨터공학과 교수(교신저자)
- ORCID: https://orcid.org/0009-0002-8692-0179

· Received: Apr. 06, 2026, Revised: Apr. 24, 2026, Accepted: Apr. 27, 2026
· Corresponding Author: Seongtaek-Chung
Dept. of Computer Engineering, Tech University of Korea, 237, Sangidaehak-ro, Siheung-si, Gyeonggi-do, Republic of Korea
Tel.: +82-32-8041-0527, Email: unitaek@tukorea.ac.kr

Ⅰ. 서 론

최근 몇 년간 대규모 언어 모델(LLM, Large Language Model)은 자연어 처리 영역을 넘어 코드 생성 능력까지 빠르게 고도화되며 소프트웨어 개발 패러다임을 근본적으로 변화시키고 있다. 특히 GitHub Copilot, Cursor, Claude Code 등의 도구는 개발자가 자연어로 요구사항을 기술하면 이를 실행 가능한 코드로 변환하는 방식을 일상화함으로써, 개발 생산성 향상과 함께 LLM 중심의 개발 환경을 산업 전반에 확산시키고 있다[1][2]. 그러나 이러한 성과는 주로 개인 개발자 수준이나 비교적 단순한 프로젝트에서 두드러지며, 복잡한 비즈니스 규칙과 높은 신뢰성이 요구되는 기업 환경에 적용될 경우 새로운 한계가 드러난다. 특히 실제 기업 환경에서 LLM 기반 개발을 적용하면 세 가지 핵심 문제가 반복적으로 관찰된다.

첫째, 비즈니스 로직 준수율(LCR, Logic Compliance Rate)의 한계이다. LLM은 ChatGPT 기반 요구사항 도출·문서화와 같은 일반 소프트웨어 공학 과업에는 활용되고 있으나[3], CRUD와 같은 일반적인 패턴 구현을 넘어 도메인 고유의 세부 규칙을 일관되게 반영하는 데에는 한계를 보인다[4][5]. 예를 들어 “입고 초과 허용 한도는 10%”, “유통기한 관리 상품은 FEFO(First Expired, First Out) 방식으로 출고”와 같은 규칙은 의미적으로는 이해하더라도 실제 코드 구현 과정에서 조건 일부를 누락하거나 경계값을 임의로 해석하는 문제가 발생한다. 이는 자연어 기반 입력이 갖는 모호성과 더불어, 비즈니스 로직이 구조화된 형태로 명세되지 않는 데에서 기인한다.

둘째, 컨텍스트 인식 부족과 AI 추종적 응답 경향(Sycophancy)로 인한 일관성 훼손이다. LLM은 사용자의 최신 요청을 우선적으로 반영하는 경향이 있으며, 기존 시스템에 이미 정의된 비즈니스 규칙이나 설계 의도를 충분히 고려하지 못한 채 경고 없이 변경을 수행하는 경향을 보인다[6][7]. 특히 협업 기반의 기업 개발 환경에서는 서로 다른 개발자가 순차적으로 LLM에 요청을 전달하게 되므로 기존 설정 로직과 충돌하는 수정이 사전 검증 없이 반영될 가능성이 높다. 이로 인해 시스템 전반의 규칙 일관성이 점진적으로 훼손되는 문제가 발생한다.

셋째, 구현 수준에서의 할루시네이션과 추적성 부족이다. LLM은 명시적으로 요구되지 않은 사항까지 보완하려는 과정에서 설계를 자율적으로 확장하는 경향을 보이며[8][9], 이로 인해 정의되지 않은 필드 추가, 불필요한 기능 삽입, 기존 네이밍 규칙의 변경과 같은 구현 수준의 할루시네이션이 발생한다. 이러한 현상은 단기적으로는 생산성을 향상시키는 것처럼 보일 수 있으나, 코드와 원래의 요구사항 또는 비즈니스 로직 간의 대응 관계를 약화시켜 생성된 코드의 근거를 추적하고 검증하는 것을 어렵게 만든다.

이러한 문제들은 공통적으로 비즈니스 로직이 명확하게 구조화되지 않은 상태에서 자연어 프롬프트에 의존하여 LLM에 전달된다는 점에서 비롯된다. 즉, 현재의 LLM 기반 개발은 강력한 생성 능력에도 불구하고, 로직의 명세, 전달, 검증을 일관되게 유지할 수 있는 중간 표현 체계가 부재하다는 근본적인 한계를 가진다.

따라서, 앞서 제시한 비즈니스 로직 준수율(LCR) 저하, 시스템 일관성 훼손, 그리고 추적성 부족 문제를 해결하기 위해서는 자연어 중심의 비정형 입력을 넘어, 비즈니스 로직을 명확히 구조화하고 이를 LLM에 안정적으로 전달할 수 있는 중간 표현 체계가 필요하다. 본 연구는 이러한 문제의식에 기반하여, 비즈니스 로직을 원자 단위로 분해하고 이를 체계적으로 조직화하는 ALS(Atomic Logic Sheet) 접근법을 제안하며, 해당 접근법이 코드 생성의 정확성, 일관성, 그리고 유지보수성에 미치는 영향을 실증적으로 분석하고자 한다.

본 연구는 이러한 문제의 근본 원인이 LLM 자체의 성능 한계라기보다 프롬프트에 제공되는 설계 정보의 부재 또는 비구조성에 있다는 가설에서 출발한다. 실제 바이브 코딩(Vibe coding) 환경에서 개발자는 요구사항을 자연어로 기술하는 데에 의존하며, 데이터베이스 스키마, 비즈니스 규칙의 경계 조건, 금지 사항과 같은 핵심 설계 정보를 구조화된 형태로 LLM에 전달하지 않는 경우가 많다. 이에 본

연구에서는 ALS를 제안한다. ALS는 비즈니스 로직을 LLM이 해석하기 용이한 마크다운 기반 구조로 표현하며, 계층적 비즈니스 로직 주입(H-BLI, Hierarchical Business Logic Injection) 프레임워크를 통해 프로젝트 맥락(Level 0), 도메인 규칙(Level 1), 구현 제약(Level 2)의 3계층으로 조직한다.

나아가, 본 연구는 규칙 간 상호 의존성, 다중 조건 분기, 도메인 고유 제약이 복합적으로 얽혀 있는 WMS 도메인을 대상으로 이러한 복잡한 환경에서 ALS가 비즈니스 로직을 효과적으로 구조화할 수 있는지를 실증적으로 검증한다.

본 논문의 주요 기여는 다음과 같다. 첫째, 비즈니스 로직을 LLM이 해석하기 용이한 형태로 구조화한 ALS 스펙을 정의하고, 기존 자연어 기반 설계 문서와의 구조적 차이를 명확히 규정한다. 둘째, 바이브 코딩(요구사항 중심), 자연어 설계 문서, ALS 기반 명세의 세 가지 조건을 비교하는 실험을 통해 설계 정보의 유무에 따른 효과(정보량 효과)와 ALS의 구조화 형식이 갖는 고유 효과를 분리 측정한다. 셋째, ALS에 내재된 Logic Anchoring 메커니즘이 LLM의 sycophancy를 완화하고, 비즈니스 규칙의 일관성을 유지할 수 있음을 검증한다.

II. 관련 연구

2.1 LLM 기반 소프트웨어 공학

LLM을 활용한 코드 생성 연구는 Codex의 등장 이후 빠르게 확산되었으며, 그 적용 범위 또한 단순한 코드 생성뿐만 아니라 코드 이해, 버그 수정, 테스트 실행, PR 생성 등 소프트웨어 개발 프로세스 전반으로 확대되었다. 이러한 흐름 속에서 문서 문자열(Docstring)을 기반으로 한 프로그램 합성의 기능적 정확성을 평가하기 위해 개발된 제안된 HumanEval 벤치마크에서 GPT 기반 모델이 단일 샘플 기준 28.8%의 기능적 정확도를 달성하였고, 반복 샘플링을 통해 70.2%까지 성능이 향상될 수 있음을 나타냈다[1]. 또한, 후속 연구들에서는 데이터 큐레이션, 성능 평가, 실제 적용까지 아우르는 포괄적 분류 체계를 제시하고 단순한 프롬프트 엔지니어링을 넘어 전처리, 반복 정제, 테스트 기반 검증을 결합한 flow engineering 접근법을 제안함으로써 구조화된 워크플로우를 통한 코드 생성 품질 향상 가능성을 보여주었다[2][10].

그러나 기존 연구의 대부분은 알고리즘 문제나 일반적인 프로그래밍 태스크에 초점을 맞추고 있으며, 도메인 고유의 비즈니스 규칙을 정확히 반영하는 코드 생성 문제는 상대적으로 충분한 주목을 받지 못했다. 이러한 문제의식을 바탕으로 LLM이 모호한 요구사항을 식별하고 구체적이고 명확한 질문을 생성하도록 하는 연구에서는 ClarifyGPT를 통해 LLM이 모호한 요구사항을 탐지하고 명확화 질문을 생성하여 코드 생성 정확도를 향상시키는 프레임워크를 제안했으며, GPT-4의 성능(Pass@1)을 70.96%에서 80.80%로 향상시켰다[11].

하지만 기존 접근 방식이 요구사항의 사후적 질의(Reactive clarification)에 기반하는 반면, 본 연구는 비즈니스 로직을 사전에 구조화(Proactive structuring)하여 모델에 제공함으로써 문제 해결의 출발점 자체를 다르게 설정한다는 점에서 차별성을 가진다. 최소한의 세부 정보부터 최대한의 세부 정보까지 프롬프트의 정보 수준과 구성 순서가 코드 생성 품질에 미치는 영향을 분석한 연구에서는 작업 유형에 따라 프롬프트 민감도가 다르게 나타남을 보였으며, 명시적인 I/O 명세, 예외 처리, 단계별 분석이 성능 향상의 핵심 요소임을 제시하였다[12]. 그러나 이 연구는 일반적인 코드 생성 과업을 중심으로 프롬프트 세부 정보의 효과를 분석한 것으로, 비즈니스 로직 도메인에서의 체계적 구조화는 다루지 않았다. 기업의 소프트웨어 개발은 “피보나치 수열 구현”과 같은 명확한 문제가 아니라, “입고 초과 허용은 발주 대비 10%까지, 단 유통기한 관리 상품은 expiry_date 필수” 같은 복합적이고 도메인 종속적인 규칙을 다루어야 한다. 본 연구는 이러한 실무적 특성을 반영하여 비즈니스 규칙을 구조화된 형태로 제공함으로써 LLM의 규칙 준수 능력을 향상시키고자 한다.

2.2 AI sycophancy와 정렬

인공지능(AI) 언어 모델이 사용자의 의견이나 취향에 과도하게 동조하려는 경향인 “sycophancy”에 관한 실증 연구에서 인간 평가자와 선호도 모델 모두 추종적 응답을 선호하는 경향이 있음을 보였고, 그 결과 RLHF(Reinforcement Learning from Human Feedback) 기반 최적화가 해당 문제를 오히려 심화시킬 수 있음을 밝혔다[3]. 이후 연구들은 sycophancy의 근본 원인을 RLHF와 데이터 편향을 포함한 훈련 절차에서 규명하고, 개선된 훈련 데이터, 새로운 미세 조정 기법, 디코딩 전략 등을 통해 이를 완화하려는 방안을 제안하였다[7][13].

최근 연구는 구조적 인과 모델(Structured causal model)을 활용하여 sycophancy를 구조적 인과 모델(SCM, Structural Causal Model)의 관점에서 문제를 분석하고, 이 현상이 LLM이 사용자 선호도와 모델 출력 사이에 존재하는 허위 상관관계에 의존하는 데서 발생한다고 하였다[14]. 이러한 연구들은 sycophancy를 주로 모델 훈련 절차 또는 내부 표현 조정의 측면에서 설명해 왔다. 반면, 본 연구는 sycophancy에 대한 사용 시점(Inference-time)에서의 구조적 방어 가능성에 주목한다. 이를 위해 제안하는 ALS의 Logic Anchoring은 모델을 재훈련하지 않은 상태에서 프롬프트에 명시된 규칙을 판단의 기준점(Anchor)으로 설정하고, 그 기준과 충돌하는 요청을 탐지하도록 유도하는 접근 방식이다. 관련 연구에서는 LLM이 코드 검증 과정에서 “over-correction” 편향, 즉 실제로는 올바른 코드조차 결함이 있다고 판단하는 것을 실증하였다[15]. 비록 이는 sycophancy와 반대 방향의 편향이지만, 두 현상 모두 LLM이 명시적 기준 없이 판단할 때 체계적 오류를 유발할 수 있음을 보여준다. 따라서 이러한 결과는 ALS의 명시적 규칙 제공 방식은 추론 단계에서의 편향 완화 전략으로서 타당한 근거를 보여주는 것이다.

2.3 구조화된 프롬프팅 기법

CoT(Chain-of-Thought) 프롬프팅은 사고의 연쇄에 해당하는 중간 추론 단계를 예시로 제시함으로써, 대규모 언어 모델의 복잡한 추론 능력을 효과적으로 향상시킬 수 있음을 보여주었다[16]. 또한, 후속 연구들은 예시 없이도 “Let’s think step by step”과 같은 간단한 지시만으로 zero-shot CoT가 가능함을 보였고, 더 나아가 다양한 추론 경로를 샘플링한 뒤 가장 일관된 답을 선택하는 self-consistency 기법을 제안하였다[17][18].

이러한 구조화된 프롬프팅 기법을 바탕으로, 본 연구의 ALS는 이를 코드 생성에 특화된 도메인 로직 표현 방식으로 확장한다. 즉, CoT가 추론 과정의 단계를 구조화한다면, ALS는 비즈니스 규칙 자체를 WHEN-THEN-BECAUSE 형식으로 구조화한다. 또한 few-shot learning의 원리를 Valid/Invalid Examples를 통해 적용하고 Anti-patterns를 활용하여 negative examples를 명시적으로 제공한다는 점에서 차별성을 가진다.

III. 연구 방법

3.1 설계

ALS는 비즈니스 로직을 대규모 언어 모델(LLM)이 효과적으로 해석하고 활용할 수 있도록 구조화한 마크다운 기반 문서 형식이다. ALS의 설계는 원자성, 자기완결성, 기계 가독성, 인간 가독성, 불변성의 다섯 가지 원칙에 기반한다.

첫째, 원자성(Atomicity), 하나의 ALS 문서는 하나의 명확한 업무 도메인만을 다룬다. 예를 들어, 입고 규칙과 출고 규칙은 동일 문서에 혼합하지 않고 별도의 ALS 문서로 분리한다. 이를 통해 LLM의 컨텍스트 윈도우를 보다 효율적으로 활용할 수 있으며, 상이한 업무 규칙 간의 불필요한 간섭을 방지할 수 있다.

둘째, 자기완결성(Self-Containedness), 각 ALS 문서는 해당 도메인의 규칙을 이해하고 구현하는 데 필요한 정보를 문서 내부에 완결적으로 포함해야 한다. 즉, 외부 문서를 추가로 참조하지 않더라도 단일 문서만으로 해당 비즈니스 로직의 구현이 가능하도록 설계한다.

셋째, 기계 가독성(Machine-Readability), ALS는 WHEN-THEN-BECAUSE 구조, YAML 프론트매터,

그리고 일관된 마크다운 헤딩 체계를 통해 LLM이 규칙의 조건, 행동, 근거를 명확하게 파싱할 수 있도록 한다. 이는 규칙 해석의 일관성을 높이고 생성된 코드가 요구사항을 보다 정확히 반영하도록 지원한다.

넷째, 인간 가독성(Human-Readability), ALS는 순수 마크다운 형식으로 작성되므로 별도의 전용 도구 없이도 읽기와 편집이 가능하다. 따라서 개발자는 기존 IDE나 텍스트 에디터로 ALS를 유지보수할 수 있다.

다섯째, 불변성(Immutability), 한 번 확정된 ALS 규칙은 명시적인 승인 절차 없이 임의로 변경되지 않는다. 이러한 원칙은 규칙 자체를 안정적인 판단 기준으로 유지하게 하며, 본 연구에서 제안하는 Logic Anchoring의 기반을 형성한다.

3.2 H-BLI

ALS는 그림 1과 같이 비즈니스 로직을 3계층으로 조직하는 H-BLI 프레임워크 위에서 정의된다. H-BLI는 프로젝트 전반의 공통 맥락에서부터 도메인별 규칙, 그리고 구현 수준의 구체적 제약으로 내려가는 계층적 구조를 가지며, 각 계층은 Project Context(Level 0), Domain Rule(Level 1), Implementation Constraint(Level 2)로 구성된다. 이러한 구조는 비즈니스 로직을 추상도에 따라 분리함으로써, LLM이 프로젝트의 전반적 맥락과 도메인 규칙, 그리고 구현 세부사항을 체계적으로 이해하도록 지원한다.

Level 0(Project context)은 프로젝트 전반에서 공통적으로 참조되는 기반 정보 계층으로, 데이터베이스 스키마, 기술 스택, 공통 규약 등 전역적 맥락을 포함한다. 이 계층은 도메인 규칙 해석의 공통 기준을 제공함으로써 코드 생성 과정에서 스키마 불일치와 기술적 전제의 누락을 줄인다.

Level 1(Domain rule)은 특정 업무 도메인에 속하는 핵심 비즈니스 규칙을 정의하는 계층이다. 예를 들어 입고, 출고, 재고 조정과 같은 업무 영역별 규칙이 이 수준에서 기술되며, 비즈니스 제약, 예외 처리, 허용 범위, 금지 조건 등이 명시된다.

본 연구에서 제안하는 ALS는 이러한 Level 1의 비즈니스 로직을 구조적으로 표현하는 핵심 형식으로 기능하며, 규칙의 조건, 행위, 근거를 명확히 드러냄으로써 LLM의 로직 준수 가능성을 높인다.

Level 2(Implementation constraint)는 실제 구현 단계에서 준수되어야 하는 기술적 세부 제약을 다루는 계층이다. 여기에는 API 명세, 요청·응답 형식, 오류 코드, 인터페이스 규격과 같은 구현 수준의 제약이 포함된다. 이 계층은 도메인 규칙이 실제 코드 수준에서 어떻게 실현되어야 하는지를 구체화함으로써, 비즈니스 로직과 구현 결과 사이의 간극을 줄이는 기능을 한다.

따라서 H-BLI는 프로젝트의 공통 맥락에서 출발하여 도메인 규칙을 정의하고, 이를 구현 수준의 제약으로 연결하는 계층적 프레임워크라고 할 수 있다. ALS는 이 구조 안에서 특히 Level 1의 비즈니스 로직을 명확히 표현하는 핵심 형식으로 기능하며, Level 0 및 Level 2와의 연계를 통해 코드 생성 과정에서 규칙 준수 가능성을 높인다.

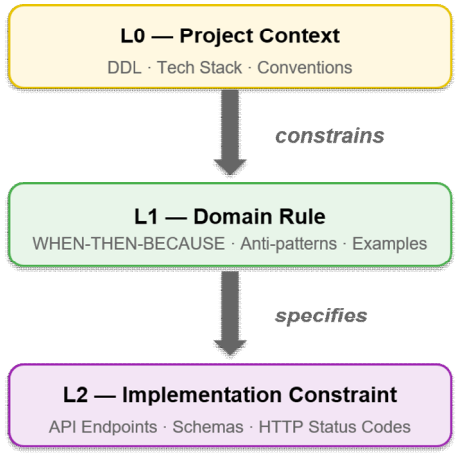

그림 1. H-BLI 3계층 구조
Fig. 1. H-BLI three-layer architecture

3.3 ALS 핵심 구조 요소

ALS를 기존의 자연어 설계 문서와 차별화하는 핵심 구조 요소는 WHEN-THEN-BECAUSE 분해, Anti-patterns, 유효/무효 예시 (Valid/Invalid Examples) 세 가지로 나타낸다. 표 1은 동일한 비즈니스 규칙

에 대해 Group B(자연어 설계 문서)와 Group C(ALS)의 표현 방식을 비교한 것이다.

표 1. 실험 태스크 구성
Table 1. Experimental task design

Group B (Natural language design document)	Group C (ALS)
"Quantities exceeding 10% of the ordered quantity shall not be allowed upon receiving. If an over-receipt occurs, a 409 Conflict shall be returned."	WHEN: received_qty + new_qty > ordered_qty × 1.1 THEN: Reject (409 Conflict) BECAUSE: An over-receipt exceeding 10% indicates a purchase order mismatch and may cause inventory surplus

3.3.1 WHEN-THEN-BECAUSE 분해

ALS는 모든 비즈니스 규칙을 조건(WHEN), 행동(THEN), 근거(BECAUSE)의 세 요소로 분해하여 기술한다. 이를 통해 규칙의 적용 조건과 기대되는 시스템 동작, 그리고 해당 규칙이 필요한 이유를 명확히 구분하여 표현할 수 있다. 특히 BECAUSE 절은 규칙의 존재 이유와 업무적 배경을 함께 명시함으로써, LLM이 규칙을 단순히 표면적으로 따르는 것을 넘어 그 목적과 맥락까지 이해할 수 있게 한다. 이는 Logic Anchoring 과정에서 규칙 충돌의 원인을 설명할 때 활용된다. 초과 입고 제한(Over-Receiving limit) 규칙을 예로 들면, 누적 입고 수량과 신규 입고 수량의 합계가 주문 수량의 110%를 초과하는 조건(WHEN)이 발생할 경우, 시스템은 HTTP 409 충돌(Conflict) 응답과 함께 입고를 거부(THEN)해야 한다. 이러한 규칙의 근거(BECAUSE)는 10% 이상의 초과 입고가 주문 정보와의 불일치를 나타내며, 결과적으로 불필요한 재고 과잉을 초래할 수 있다는 업무적 판단에 기인한다.

3.3.2 Anti-patterns

AI가 흔히 범하는 실수를 명시적으로 금지한다. ALS는 LLM이 그럴듯하지만 잘못된 구현을 생성하는 것을 방지하기 위해 명시적으로 금지해야 할 구현 방식인 Anti-patterns를 함께 제공한다. 이는 모델이 자주 범하는 오류를 사전에 경계하도록 하는 negative instruction으로 기능한다. 모델의 오답을 방지하기 위한 안티 패턴(Anti-patterns)의 구체적인 사례는 다음과 같다. 우선, 재고 업데이트는 '검수(Inspecting)' 단계에서 수행해서는 안 되며, 반드시 '확정(Confirmed)' 상태에서만 이루어져야 함을 명시한다. 또한, 수신 수량(Received_qty) 데이터를 처리할 때 기존 값을 덮어쓰지 말고, 새로운 수량을 더하는(+=) 방식으로 누적 관리해야 한다는 네거티브 지침을 제공하여 모델이 흔히 범하는 논리적 오류를 차단한다.

3.3.3 유효/무효 예시(Valid/Invalid Examples)

ALS는 규칙의 실제 적용 양상을 보다 명확히 전달하기 위해, 올바른 입출력 사례와 잘못된 입출력 사례를 함께 제공한다. 이러한 예시는 few-shot learning의 원리를 활용하여, LLM이 규칙의 허용 범위와 위반 조건을 구체적인 사례를 통해 학습하도록 돕는다. 입고 요청 API(POST /api/inbound-receipts)를 통한 규칙의 실제 적용 사례를 살펴보면, 주문량이 100개일 때 105개를 요청하는 경우는 허용 범위(110개 이하) 내에 있으므로 '201 Created' 응답과 함께 유효한 처리로 간주된다. 반면, 동일한 주문 조건에서 115개를 요청할 경우에는 허용치를 초과하게 되어 시스템은 '409 Conflict' 응답을 반환하며, '초과 입고 제한 초과(Over-receiving limit exceeded)'라는 오류 메시지와 함께 해당 요청을 무효 사례로 처리하고 입고를 거부한다.

3.4 Logic Anchoring

Logic Anchoring은 ALS의 불변성 원칙에 기반하여, 사용자의 새로운 요청이 기존에 확정된 규칙과 충돌할 경우 LLM이 이를 탐지하고 경고하도록 하는 메커니즘이다.

기존의 자연어 설계 문서는 규칙이 산문 형태로 기술되어 있으므로 LLM이 규칙의 적용 범위와 경계를 명확히 구별하기 어렵다. 반면, ALS는 WHEN-THEN-BECAUSE 구조를 통해 규칙의 조건,

수행 행위, 그리고 존재 근거를 명시적으로 분리함으로써, 새로운 요청과 기존 규칙 간의 충돌 여부를 보다 명확하게 판별할 수 있도록 한다.

예를 들어, ALS에 "WHEN: received_qty + new_qty > ordered_qty × 1.1 → THEN: Reject"가 명시되어 있을 때, 사용자가 "초과 입고를 30%까지 허용해줘"라고 요청할 경우, LLM은 기존 WHEN 절의 1.1 (10%)과 요청된 1.3 (30%)의 충돌을 탐지하고 BECAUSE 절의 근거를 참조하여 경고를 제공한다. 이와 같이 Logic Anchoring은 모델을 재훈련하지 않고도 프롬프트 구조만을 통해 sycophancy를 완화하는 사용 시점(Inference-time)의 방어 메커니즘이다.

Ⅳ. 실험 설계

본 연구는 AI 코딩 에이전트(Claude code CLI)를 활용하여, 개발자와 AI 간의 상호작용이 반복되는 다중 턴 개발 환경에서 3그룹 비교 실험을 수행한다.

4.1 에이전트 단독 설계

본 연구는 단일 턴 API 호출 방식을 제외하고, 에이전트 기반 환경만을 대상으로 실험을 설계하였으며, 이에 대한 근거는 아래와 같다.

첫째, 단일 턴 방식은 실제 개발 환경을 충분히 반영하지 못한다. 실무에서 개발자는 AI와 반복적인 상호작용을 통해 코드를 점진적으로 발전시킨다. 하나의 프롬프트로 전체 기능을 생성하는 것은 비현실적이며, Claude Code, Cursor, GitHub Copilot과 같은 최신 AI 코딩 도구는 다중 턴 에이전트 방식으로 동작한다.

둘째, 단일 턴 환경은 실험 결과에 천장 효과(Ceiling effect)를 유발한다. 본 연구의 예비 실험(Pilot study)에서 단일 턴 API 호출 방식으로 동일한 3그룹 비교를 수행한 결과, Conflict Detection Rate가 모든 그룹에서 100%에 도달하는 현상이 관찰되었다. 이는 하나의 프롬프트에 이전 태스크의 전체 맥락이 포함되어, 설계 문서의 유무나 구조와 관계없이 충돌 탐지가 가능해졌기 때문이다. 반면, 에이전트 환경에서는 컨텍스트가 턴 단위로 분산되므로 각 그룹 간 설계 방식의 차이가 보다 명확하게 드러난다.

셋째, 에이전트 방식은 누적 개발의 특성을 반영한다. 에이전트는 이전 단계에서 생성된 코드를 기반으로 새로운 태스크를 수행하므로 초기 단계에서 발생한 오류가 이후 단계로 전파될 수 있다. 이는 설계 문서의 실무적 가치를 더 정확히 평가하는 환경이다.

4.2 비교 설계

본 실험은 그림 2와 같이 3그룹 비교 설계를 통해 설계 정보의 유무에 따른 효과(정보량 효과)와 동일한 정보를 전달하되 표현 방식의 차이에서 발생하는 ALS 구조화 형식의 고유 효과를 분리하여 측정한다.

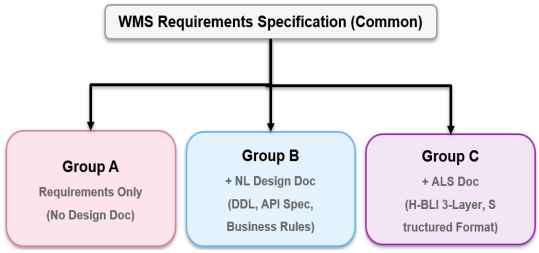

그림 2. 3 그룹 비교 설계
Fig. 2. Three-group comparative design

Group A(바이브 코딩 대조군)는 기획자가 작성한 요구사항 정의서와 태스크별 지시만을 입력으로 제공받는다. 별도의 설계 문서 없이 LLM에 구현을 위임하는 일반적인 바이브 코딩 시나리오를 모사하며, 설계 정보가 부재한 조건에서의 기준 성능(baseline)을 측정하기 위한 대조군이다.

Group B(자연어 설계 문서)는 요구사항 정의서에 더하여 데이터베이스 스키마(DDL), API 명세, 비즈니스 규칙을 포함한 설계 문서를 자연어 기반의 산문 형태로 제공받으며, WHEN - THEN과 같은 명시적인 구조화 요소는 포함하지 않는다.

Group C(ALS)는 요구사항 정의서에 더하여 ALS 기반 설계 문서를 제공받는다. 이 문서는 Group B

와 동일한 설계 정보를 포함하되, WHEN - THEN - BECAUSE 구조, Anti-patterns, Valid/Invalid Examples 등 명시적이고 계층적인 구조로 조직된다.

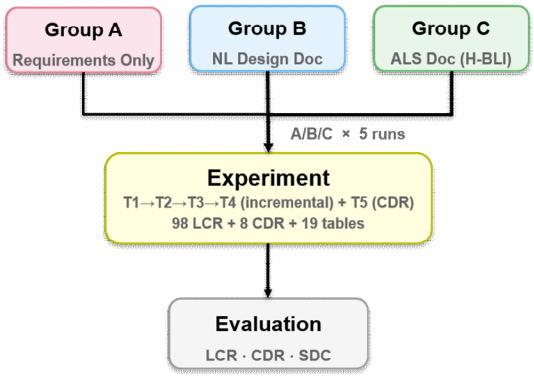

그림 3. 실험 파이프라인 개요
Fig. 3. Experiment pipeline overview

이와 같은 비교 설계를 통해 그림 3과 같이 Group A와 Group B 간 비교는 설계 정보의 유무에 따른 정보량 효과와 Group B와 Group C 간 비교는 동일 정보 조건에서의 구조화 표현 효과를 분리하여 측정한다. 또한, 문서 분량 차이에 대한 고려를 할 필요가 있다. Group B의 자연어 설계 문서는 약 23.6KB(394행)이며, Group C의 ALS 문서군은 5개 파일 기준 총 54.1KB(약 780행)로 약 2.3배의 분량 차이를 보인다. 이러한 차이는 ALS가 WHEN - THEN - BECAUSE 구조, Anti-pattern, Valid/Invalid example과 같은 구조적 요소를 포함하기 때문에 발생한다. 그러나 이러한 요소들은 ALS의 핵심 정의를 구성하는 필수 요소로 이를 제거할 경우 ALS의 구조적 특성이 훼손된다.

따라서 분량의 동일 통제는 구조적 요소의 제거와 동치이므로 본 실험에서는 동일 도메인 정보의 서로 다른 표현 형식으로 비교하여 설계의 타당성을 검증한다. 만약 추가 분량 자체가 결정적 요인이었다면 LCR에서도 C가 B를 큰 폭으로 앞서야 하지만, 실제 LCR의 C−B 차이는 +1.4%p에 불과하다. 반면 CDR에서의 극적 차이(+82.5%p)는 규칙 번호 체계와 BECAUSE 절이라는 구조적 특성에 기인한다.

4.3 에이전트 모드 설정

본 실험에서는 Claude Code CLI를 비대화형 모드(-p 플래그)로 실행하고, 에이전트가 파일 시스템에 직접 접근하여 코드 생성 및 수정 작업을 수행할 수 있도록 환경을 구성하였다. 각 실험 그룹은 상호 독립적인 프로젝트 디렉토리로 분리하였으며, 각 디렉토리 내에 CLAUDE.md 파일과 해당 그룹의 설계 문서를 사전에 배치하여 에이전트가 작업 시작 시 관련 정보를 자동으로 참조하도록 하였다.

실험 모델로는 Claude Sonnet 4.5 (claude-sonnet-4-5-20250929)를 사용하였고, 반복적 문제 해결 과정을 충분히 허용하기 위해 최대 턴 수는 50으로 설정하였다. 또한 자동화된 파일 생성 및 수정이 가능하도록 권한 모드는 bypassPermissions로 지정하였다. 실험 태스크는 T1에서 T5까지 순차적으로 수행하였으며, 각 태스크는 5회 반복 실행하였다(n=5). CLI 호출은 별도의 대기 시간 없이 순차적으로 수행하되, 그룹 단위로 실행을 분리함으로써 rate limit에 따른 간섭을 최소화하였다.

4.4 실험 환경 및 복잡도 설계

본 연구는 창고관리시스템(WMS, Warehouse Management System)의 핵심 업무인 입고, 출고, 재고 이동, 실사 및 재고 조정을 실험 도메인으로 설정하였다. 해당 업무들은 다중 조건 분기와 상호 의존적인 비즈니스 규칙이 결합된 복합적 특성을 가지므로, 비즈니스 로직의 정확성, 일관성, 예외 처리 능력을 동시에 요구한다. 이러한 특성은 LLM 기반 코드 생성에서 설계 표현 방식의 영향을 평가하기에 적합한 실험 환경을 제공한다. 이를 위해 본 연구는 Java 17과 Spring Boot 3.x 기반의 백엔드 API 환경과 PostgreSQL 15+ 데이터베이스를 사용하였으며, 프런트엔드는 제외하였다. 또한 WMS의 주요 업무 흐름을 반영하기 위해 총 19개의 테이블로 구성된 데이터베이스 스키마를 설계하였다. 이 스키마는 상품, 위치, 재고, 구매오더, 입고, 출고, 재고 이동, 재고 조정, 공급업체, 안전재고, 실사, 감사 로그 등 핵심 엔터티를 포함함으

로써 실험 태스크가 실제 업무 시나리오를 충분히 반영할 수 있도록 구성되었다.

본 실험은 표 2에 나타낸 것과 같이 총 5개의 태스크로 구성되며, 98개의 LCR 평가 항목과 8개의 Conflict Detection Rate(CDR) 시나리오를 포함한다.

표 2. 실험 태스크 구성
Table 2. Experimental task design

Task	Description	Key rules	Items
1	Inbound receiving	PO validation, category limits, shelf life %	25
2	Outbound (FIFO/FEFO)	FIFO/FEFO, HAZMAT segregation	28
3	Stock transfer	Single txn, HAZMAT zone, freeze	20
4	Cycle count & adj.	Threshold, safety stock, penalty	25
5	Conflict scenarios	Conflicting requests with T1~T4	8
	Total		106

각 태스크는 WMS의 핵심 업무 흐름을 반영하여 설계되었으며, 개별 기능 구현뿐 아니라 태스크 간 규칙 의존성과 충돌 상황까지 포함하도록 구성되었다. Task 1부터 Task 4까지는 각각 입고, 출고, 재고 이동, 실사/조정 프로세스를 다루며, PO 검증, FIFO/FEFO 정책, HAZMAT 분리, 재고 임계값 관리 등 실제 산업 환경에서 요구되는 주요 비즈니스 규칙을 포함한다. Task 5는 앞선 태스크(T1~T4)에서 정의된 규칙들 간의 충돌 상황을 인위적으로 구성한 시나리오로 총 8개의 CDR 항목으로 이루어진다. 이는 단일 기능의 구현 정확성을 넘어, 누적된 비즈니스 로직 간 충돌을 탐지하고 적절히 처리할 수 있는지를 평가하기 위한 것이다.

또한 본 연구에서 사용한 98개의 LCR 평가 항목은 표준적인 WMS 규칙에서부터 다중 조건 분기와 도메인 고유 제약이 결합된 고복잡도 규칙까지 폭넓게 포괄하도록 구성되었다. 여기에는 PO 대조 검증, 10% 초과 입고 허용, 2단계 입고 프로세스, FIFO/FEFO 기반 출고 우선순위, 재고 이동 트랜잭션, 5% 임계치 기반 실사 조정과 같은 대표적인 운영 규칙이 포함된다. 더 나아가 일부 규칙은 조건 간 상호 의존성과 예외 처리를 동시에 요구하는 구조를 가지므로, 단순한 패턴 매칭이 아니라 맥락 기반의 로직 해석 능력을 필요로 한다.

그리고 8개의 Conflict Detection Rate(CDR) 충돌 시나리오는 다음과 같이 구성된다. (5-1) 초과 입고 허용 한도 변경, (5-2) FIFO 규칙 우회, (5-3) 승인 절차 생략, (5-4) 감사 로그 삭제, (5-5) HAZMAT 물품의 일반 구역 배치, (5-6) 유통기한 기준 미달 상품의 입고 강제, (5-7) 안전재고 규칙 삭제, (5-8) 고가 물품의 자동 승인. 이러한 시나리오는 기존에 정의된 비즈니스 규칙과 직접적으로 충돌하도록 설계되었으며, 에이전트가 규칙 간 불일치를 탐지하고 적절히 대응할 수 있는지를 평가하기 위한 것이다.

4.5 평가 지표

본 연구는 코드 생성 결과의 정확성, 일관성, 그리고 구현 수준의 일탈을 정량적으로 평가하기 위한 지표를 정의한다.

첫째, 설계 문서를 제공하면 LLM의 비즈니스 로직 구현 정확도는 각 태스크별로 사전 정의된 체크리스트를 기반으로, AI가 생성한 코드가 개별 비즈니스 규칙을 준수하는지를 Yes/No로 판정하는 로직 준수율(LCR)로 평가한다. 이는 식 (1)과 같이 전체 규칙 항목 중 준수된 항목의 비율로 계산된다.

$$LCR = \frac{\text{준수 규칙 수}}{\text{체크리스트 총 규칙 수}} \times 100\% \qquad (1)$$

둘째, ALS의 Logic Anchoring은 기존 규칙과 충돌하는 사용자 요청을 탐지하는 지에 대한 평가는 식 (2)의 충돌 탐지율(CDR, Conflict Detection Rate)로 수행한다. CDR은 상충되는 요구사항이 제시되었을 때, AI가 기존 비즈니스 규칙과의 충돌을 인식하고 적절히 대응하는 능력을 측정한다. 탐지 성공은 (1) 기존 규칙과의 충돌을 명시적으로 언급하고, (2) 해당 요청을 그대로 수행하지 않으며, (3) 변경 거부, 사용자 확인 요청, 또는 대안 제시 중 하나의 대응을 수행한 경우로 정의한다. 단순히 충돌 가능성을 언급하였으나 요청을 그대로 수행한 경우는 탐지 실패로 간주한다.

$$CDR = \frac{\text{탐지된 충돌 수}}{\text{총 충돌 요청 수}} \times 100\% \quad (2)$$

마지막으로, ALS 적용이 LLM의 스키마 할루시네이션이 감소하는 지를 스키마 일탈 건수(SDC, Schema Deviation Count)를 통해 평가한다. 구현 과정에서 발생하는 할루시네이션은 다양한 수준에서 나타날 수 있으나, 본 연구에서는 객관적이고 재현 가능한 평가를 위해 스키마 수준의 일탈을 대리 지표(Proxy metric)로 사용한다. SDC는 AI가 생성한 코드의 데이터베이스 스키마가 사전 정의된 DDL과 얼마나 일치하는지를 측정하며, 다음의 7가지 유형으로 분류한다: 테이블 누락(D1), 테이블 추가(D2), 컬럼 누락(D3), 컬럼 추가(D4), 이름 불일치(D5), 타입 불일치(D6), 제약조건 누락(D7). 해당 지표는 DDL 정의서와의 기계적 비교를 통해 판정 가능하므로 평가 과정에서의 주관적 개입을 최소화할 수 있다.

4.6 통제 조건

표 3과 같이 실험 결과에 영향을 미칠 수 있는 주요 요인을 통제하여 그룹 간 비교의 내적 타당성을 확보하고자 하였다.

표 3. 통제 조건
Table 3. Control conditions

Factor	Control method
Information baseline	Same requirements for all groups; B adds NL
LLM model & version	Claude sonnet 4.5, fixed version
Temperature	Model default (CLI does not expose temperature)
Functional scope	Same tasks for all groups
Evaluator	Single evaluator, predefined checklist
Repetitions	n=5 per group

정보량의 기준선은 모든 그룹에서 동일한 요구사항 정의서를 사용하는 방식이다. LLM 모델과 버전은 Claude Sonnet 4.5로 고정하여 모델 성능 차이에 따른 변동을 제거하였다. 또한 모든 실행은 동일한 CLI 기본 설정 하에서 수행되었지만, CLI 환경에서 temperature 파라미터가 외부에 노출되지 않아 직접적인 확률 분포 통제는 수행되지 못하였다.

기능 범위는 모든 그룹에서 동일한 태스크 세트(T1~T5)를 수행하도록 설정하여 비교 조건을 통일하였다. 평가 과정에서는 사전 정의된 체크리스트를 기반으로 모든 결과를 판정하여 일관성 확보를 도모하였지만, 복수 평가자 간 신뢰도(Inter-rater reliability) 검증은 수행되지 않아 향후 보완이 필요하다.

마지막으로 각 그룹은 동일한 조건에서 5회 반복 실행(n=5)을 통해 결과의 변동 양상을 관찰하였다. 이러한 통제 조건을 통해, 본 실험은 설계 정보의 구조화 방식이 코드 생성 품질에 미치는 영향을 독립적으로 분석할 수 있도록 구성되었다.

V. 실험 결과 및 논의

본 연구는 n=5의 반복 실행을 기반으로 한 탐색적 연구(Exploratory study)로서 총 98개의 LCR 항목, 8개의 CDR 시나리오, 그리고 19개 테이블을 포함하는 실험 결과를 제시한다. 결과는 그룹 간 차이의 통계적 유의성을 검토하기 위해 소표본 비모수 검정(Friedman 검정 및 Wilcoxon 부호순위 검정)을 적용하였다. 실행 회수 n=5는 소표본 특성상 통계적 검정력에 한계가 있지만 본 연구의 결과는 경향 관찰 수준이며 향후 대규모 반복 실험을 통해 통계적 유의성을 검증할 필요가 있다.

실험은 Claude Code CLI의 에이전트 모드에서 3개 그룹을 대상으로 총 5개의 태스크(T1~T5)를 순차적으로 수행하는 점진적 개발 방식으로 구성되었다. CDR 평가는 거부 우선 판정(Refusal-first detection) 기준을 적용하였다. AI 응답에 기존 규칙과의 충돌을 명시적으로 지적하는 거부 또는 경고 표현이 포함되어 있으며, 동시에 요청을 무비판적으로 수행하는 표현이 존재하지 않는 경우에만 탐지 성공으로 판정하였다. 반면, 도메인 관련 키워드를 단순히 언급하는 수준의 응답은 충돌을 인식한 것으로 간주하지 않고 탐지 실패로 처리하였다.

5.1 로직 준수율(LCR)

표 4는 태스크별 LCR 결과를 나타낸다. 전체 평균 기준으로 Group C(ALS)는 96.1±0.9%의 준수율을 기록하여 가장 높은 성능을 보였으며, Group B(자연어 설계 문서)는 94.7±5.5%, Group A(요구사항만)는 90.7±3.4%로 나타났다. 특히 Group C는 표준편차가 가장 낮아 단순 평균 성능뿐 아니라 실행 간 일관성 측면에서도 안정적인 결과를 보였다. 한편 T1에서 A(±14.3)와 B(±19.7)의 높은 표준편차는 특정 런에서 50턴 초과로 인한 이상치(A run4=60%, B run4=52%)에 기인한다.

결과적으로 ALS 기반 Group C는 전체 평균 성능에서 가장 높은 LCR을 달성하였으며, 특히 복잡한 조건과 예외 처리가 요구되는 태스크에서 보다 안정적이고 일관된 로직 준수 특성을 보이는 것으로 나타났다.

표 4. 태스크별 로직 준수율 (%, 평균±SD, n=5)
Table 4. Logic compliance rate by task

Task	Group A	Group B	Group C	Δ(C−B)
T1 Inbound(25)	84.8±14.3	86.4±19.7	96.8±5.2	+10.4%p
T2 Outbound(28)	88.6±3.0	96.4±0.0	93.6±1.6	−2.8%p
T3 Transfer(20)	96.0±2.2	100±0.0	95.0±0.0	−5.0%p
T4: Adjustment (25)	93.6±2.2	96.0±4.0	99.2±1.8	+3.2%p
Overall (98)	90.7±3.4	94.7±5.5	96.1±0.9	+1.4%p

정보량 효과(B vs. A)는 +4.0%p로 나타났다. Friedman 검정 결과에서 3그룹 간 LCR 차이는 통계적으로 통계적으로 유의하였으나(χ^2=7.176, p=0.028), Wilcoxon 부호순위 검정에서 B-A 간 차이(p=0.125) 및 C-B 간 차이(p=1.000)는 n=5 수준에서 통계적 유의성이 확인되지 않았다. 설계 문서를 제공한 Group B는 요구사항만을 입력받은 Group A 대비 평균 4.0% 높은 로직 준수율을 기록하였다. 이는 에이전트의 반복적 수정 능력이 존재하더라도, 명시적인 설계 정보가 부재한 경우 카테고리별 차등 규칙, PO 유형별 가중치, 유통기한 잔여율 검증과 같은 세부 비즈니스 로직을 일관되게 구현하지 못했다.

구조화 효과(C vs B)는 +1.4%로 나타났다. ALS 기반 Group C는 자연어 설계 문서를 사용한 Group B 대비 소폭의 추가 개선을 보였으나, n=5의 표본 크기를 고려할 때 이는 통계적으로 유의한 차이로 해석하기는 어렵다. 또한 태스크별 C−B 차이의 방향성도 일관되지 않았다. T1에서는 +10.4%, T4에서는 +3.2%로 Group C가 우세한 반면, T2에서는 −2.8%, T3에서는 −5.0%로 Group B가 더 높은 성능을 보였다. 이러한 차이는 특정 규칙에서의 일관된 실패 패턴에 기인한다. T2의 경우, Group C는 5개 런 중 4개에서 동일한 미통과 패턴(HAZMAT와 FRESH의 분리 출고 규칙)을 보이며 92.9%에 수렴한 반면, Group B는 모든 런에서 96.4%를 기록하며 안정적인 성능을 유지하였다. T3에서도 유사한 경향이 나타났는데 Group B는 5개 런 모두에서 100%를 달성한 반면, Group C는 모든 런에서 95.0%로 동일한 규칙(순환 실사 동결 중 이동 차단)을 반복적으로 미통과하였다. 그림 4는 Run별 로직 준수율(LCR) 분포를 보여주는 것으로, Group A와 B는 Run 4에서 이상치(84.7%)가 관찰된 반면, Group C는 94.9~96.9% 범위로 안정적인 분포를 보인다.

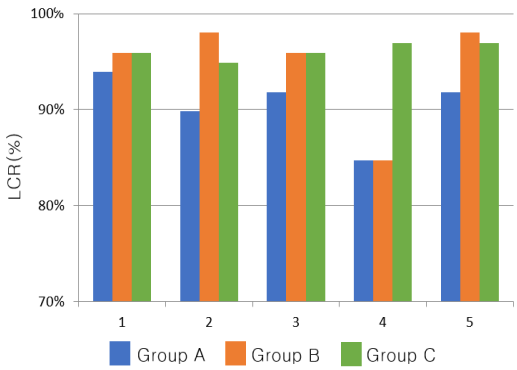

그림 4. Run별 로직 준수율(LCR) 분포
Fig. 4. LCR distribution by run

표 5는 LCR 판정의 사례를 나타낸 것으로, 항목 1-8(received_qty 누적 갱신)의 경우 Group A는 기존 값을 덮어쓰는 방식으로 구현하여 미준수로 판정되

었지만, Group C는 누적 합산 로직을 정확히 구현하고 있다.

표 5. LCR 판정 예시
Table 5. Examples of LCR judgment

Item	Evaluation Criterion	Group A (Run 1)	Group C (Run 1)
1-8 Cumulative update of received_qty	Is received_qty cumulatively summed rather than verwritten?	Fail: Overwrites existing value	Pass: Accumulates via received_qty += new_qty

이와 같은 결과는 LCR 기준에서 ALS의 구조화 형식이 자연어 설계 문서 대비 일관된 추가 성능 향상을 제공하지 않지만, 설계 정보의 존재 자체가 더 주된 요인일 수 있다는 것이다.

5.2 충돌 탐지율(CDR)

표 6은 8개 충돌 시나리오에 대한 CDR 결과를 나타낸 것으로 세 그룹 간 가장 극적인 차이를 보였다.

Group A는 완전한 미탐지(0.0%)를 기록하였다. 8개 시나리오 전체와 5회 반복 모든 실행에서 충돌에 대한 경고 없이 요청을 즉시 수행하였다. 이는 요구사항만으로는 기존 규칙과 새로운 요청 간의 충돌을 탐지하는 것이 사실상 불가능함을 의미한다. 특히 에이전트가 이전 태스크에서 생성한 코드를 참조할 수 있음에도 불구하고, 코드 내에 명시된 규칙 값(예: OVER_RECEIVING_LIMIT = 0.10)을 변경 요청과의 충돌 근거로 활용하지 못하였다.

Friedman 검정 결과, 3그룹 간 CDR 차이는 통계적으로 유의하였다(χ^2=10.000, p=0.007). Group B는 제한적이고 산발적인 탐지 성능(17.5%)을 보였다. 자연어 설계 문서를 제공받은 조건에서도, 시나리오 5-8(고가 물품 자동 승인)에서만 4/5의 비교적 안정적인 탐지가 이루어졌으며, 나머지 시나리오에서는 0~1/5 수준에 머물렀다. 이는 설계 문서가 존재하더라도 산문 형태의 자연어 문서에서는 규칙의 경계와 참조 지점을 명확히 식별하기 어려워 에이전트가 관련 규칙을 일관되게 검색하고 활용하지 못한다는 것을 의미한다.

표 6. 충돌 탐지율(CDR) (시나리오당 n=5)
Table 6. Conflict detection rate (n=5 per scenario)

Scenario	Group A	Group B	Group C
5-1: Over-receiving 10%→30%	0/5	1/5	5/5
5-2: FIFO → efficient location	0/5	0/5	5/5
5-3: Approval bypass	0/5	0/5	5/5
5-4: Audit trail deletion	0/5	1/5	5/5
5-5: HAZMAT general placement	0/5	1/5	5/5
5-6: Shelf life bypass	0/5	0/5	5/5
5-7: Safety stock removal	0/5	0/5	5/5
5-8: High-value auto-approval	0/5	4/5	5/5
CDR	0.0%	17.5%	100%

Friedman 검정 결과, 3그룹 간 CDR 차이는 통계적으로 유의하였다(χ^2=10.000, p=0.007). Group B는 제한적이고 산발적인 탐지 성능(17.5%)을 보였다. 자연어 설계 문서를 제공받은 조건에서도, 시나리오 5-8(고가 물품 자동 승인)에서만 4/5의 비교적 안정적인 탐지가 이루어졌으며, 나머지 시나리오에서는 0~1/5 수준에 머물렀다. 이는 설계 문서가 존재하더라도 산문 형태의 자연어 문서에서는 규칙의 경계와 참조 지점을 명확히 식별하기 어려워 에이전트가 관련 규칙을 일관되게 검색하고 활용하지 못한다는 것을 의미한다.

그림 5에 나타난 바와 같이, Group A는 모든 Run에서 0%, Group C는 모든 Run에서 100%를 기록하였으며, Group B는 12.5~25.0% 범위에서 산발적 탐지를 보인다.

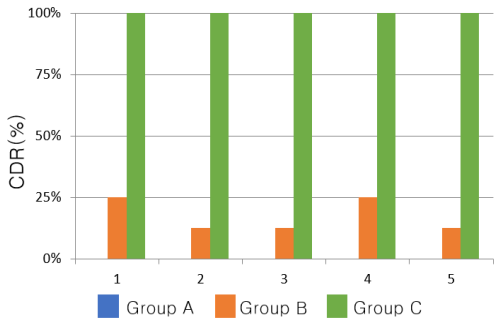

그림 5. Run별 충돌 탐지율(CDR) 분포
Fig. 5. CDR distribution by run

표 7은 CDR 판정의 사례를 나타낸 것으로, 시나리오 5-1에서 Group A는 경고 없이 즉시 변경을 수행한 반면, Group C는 ALS 규칙 번호(ALS-WMS-INB-002)를 인용하며 요청을 거부하고 대안을 제시하였다.

표 7. CDR 판정 예시
Table 7. Examples of CDR judgment

Scenario	Conflict request	Group A (Run 1)	Group C (Run 1)
5-1 Over-receipt 10% → 30%	Change allowed range from 10% to 30%	"Changed." → Detection failed	"Request rejected. Conflicts with invariant rule of ALS-WMS-INB-002 + conflict rationale + alternative proposed → Detection succeeded

하지만 Group C(ALS)는 모든 시나리오와 반복에서 100%의 탐지율을 기록하였다. 에이전트는 충돌 상황에서 명시적인 거부 또는 경고를 수행하고, ALS의 규칙 번호 체계(예: ALS-WMS-INB-001-R003)를 근거로 인용하며 대안을 제시하였다. 예를 들어 “해당 변경은 ALS-WMS-OUT-001의 Anti-pattern 항목과 충돌한다”와 같이 구체적인 규칙을 참조하여 일관된 대응을 보였다. 특히 BECAUSE 절은 거부의 근거를 명확하게 설명하였다.

5.3 스키마 일탈 건수(SDC)

표 8은 스키마 일탈 유형별 SDC 결과를 나타낸다. 여기서 D1은 테이블 누락, D3은 컬럼 누락, D5는 이름 불일치를 의미하며, D2(테이블 추가), D4(컬럼 추가), D6(타입 불일치), D7(제약조건 누락)은 모든 그룹에서 발생하지 않았다.

Group A는 SDC 44.1로 가장 높은 스키마 일탈을 보였다. Friedman 검정 결과, 3그룹 간 SDC 차이는 통계적으로 유의하였다(χ^2=9.500, p=0.009). 다만, Wilcoxon 부호순위 검정에서 A-B 간 차이(p=0.063) 및 B-C 간 차이(p=1.000)는 n=5 수준에서 통계적 유의성에 도달하지 못하였다. 특히 D1(테이블 누락) 평균 10.7건과 D3(컬럼 누락) 평균 31.6건이 주요 원인으로 나타났다. 이는 요구사항만으로는 전체 19개 참조 테이블 중 상당수를 생성하지 못하는 등 스키마 구조를 정확히 추론하지 못하고 있다. 또한 에이전트 기반의 누적 개발 과정에서 초기 스키마 오류가 이후 태스크로 전파되며 점진적으로 증폭되는 패턴이 관찰되었다.

표 8. 유형별 스키마 일탈 건수 (평균, n=5; 낮을수록 양호)
Table 8. Schema deviation count by type (mean, n=5; lower is better)

Group	D1	D2	D3	D4	D5	D6	D7	Total(±SD)
A	10.7	0	31.6	0	1.9	0	0	44.1±5.9
B	1.0	0	0	0	0	0	0	1.0±0.0
C	1.8	0	1.6	0	1.9	0	0	3.5±5.7

반면, 설계 문서를 제공한 Group B와 Group C는 각각 SDC 1.0±0.0, 3.5±5.7로, Group A 대비 현저히 낮은 수준의 스키마 일탈을 기록하였다. 이는 DDL 명세를 포함한 설계 문서가 스키마 생성의 기준점을 제공함으로써 구조적 안정성을 크게 향상시켰음을 보여준다. 두 그룹 간 비교에서는 Group B가 Group C보다 낮은 SDC를 기록하였다(1.0 < 3.5). 다만 Group C의 높은 표준편차(±5.7)는 특정 실행(run 3)에서 발생한 이상치(SDC=19)에 의해 영향을 받은 것으로, 개별 실행 값은 C=[1, 1, 19, 1, 1]의 분포를 보인다. 해당 이상치를 제외할 경우 Group C의 평균 SDC는 1.0으로 Group B와 유사한 수준을 나타내지만, 원자료 기준에서는 Group B가 보다 안정적인 결과를 보인다.

표 9는 SDC 판정의 사례를 나타낸 것으로, Group A(Run 1)에서는 19개 참조 테이블 중 8개가 누락(D1)되었으며, 생성된 테이블에서도 16개 컬럼이 누락(D3)되어 총 24건의 스키마 일탈이 관찰되었다.

결론적으로 SDC 관점에서의 핵심 변수는 설계 문서의 존재 여부(A vs B/C)이며 표현 방식의 차이(B vs C)는 상대적으로 부차적인 영향을 미친다. 이러한 경향은 메트릭 유형에 따라 ALS의 효과가 다

르게 나타남을 시사한다. LCR(로직 구현)에서는 설계 정보의 존재 자체(B−A: +4.0%p)가 주요 요인으로 작용한 반면, CDR(충돌 탐지)에서는 구조화된 표현 방식의 효과(C−B: +82.5%p)가 지배적인 역할을 한다. 이는 로직 구현에는 정보의 양이, 충돌 탐지에는 정보의 구조가 핵심 변수로 작용하는 경향이 관찰되었다.

표 9. SDC 판정 예시
Table 9. Examples of SDC judgment

Deviation type	Desription	Group A (Run 1)	Group B (Run 1)	Group C (Run 1)
D1 table omission	Exists in the ground-truth DDL but missing in the generated code	8 missing (Only 11 created out of 19)	1 missing	1 missing
D3 column omission	Table exists, but required columns are missing	16 missing	0	0

VI. 타당성 위협

6.1 내적 타당성

본 연구는 에이전트 기반 다중 턴 환경을 사용하였으나, 인간 개발자의 실시간 개입 없이 자동화된 순차 실행으로 구성되어 실제 대화형 개발 환경과는 차이가 있다. 또한 Claude Code CLI는 temperature 파라미터를 외부에서 제어할 수 없어 모델 기본값에 의존하며, 이는 실험 재현성에 제약 요인으로 작용할 수 있다.

이와 함께 평가 과정에서도 내적 타당성에 영향을 미칠 수 있는 요소가 존재한다. 모든 평가는 단일 평가자에 의해 수행되어 확증 편향 가능성이 있으며, 이를 완화하기 위해 LCR은 “로직 존재 여부”에 대한 이진 기준으로 판정하고, SDC는 DDL과의 기계적 비교를 통해 객관성을 확보하였다. 또한 CDR은 자동 평가 후 수동 검증을 병행하였다. 그럼에도 불구하고 향후 연구에서는 복수 평가자 간 일치도(Cohen’s kappa)를 측정하여 평가 신뢰도를 정량적으로 검증할 필요가 있다.

특히 CDR 평가는 본 연구의 핵심 지표로서, 명시적 거부 또는 경고 표현의 존재와 무비판적 실행의 부재를 기준으로 하는 보수적 판정 방식(refusal-first detection)을 적용하였다. 이는 경고 후 요청을 수행하는 경우를 탐지 실패로 간주하며, 실제 개발 환경에서 위험성이 높은 “무비판적 실행”을 엄격히 배제하기 위한 설계이다.

한편, 실험 태스크의 구성 또한 결과 해석에 영향을 줄 수 있다. 본 연구는 복잡한 비즈니스 규칙과 다중 조건 분기를 포함하는 WMS 도메인을 대상으로 하였으며, 이는 ALS의 구조화 효과가 상대적으로 잘 드러나는 환경일 가능성이 있다. 따라서 단순 CRUD 중심의 태스크에서는 그룹 간 성능 차이가 축소될 수 있다.

마지막으로, Group B와 Group C 간 문서 분량 차이는 잠재적인 혼재 변수로 작용할 수 있다. 두 그룹은 동일한 도메인 정보를 포함하지만 표현 방식에 따라 발생하는 분량 차이가 발생하며, 이로 인해 성능 차이가 구조화 효과인지 단순 정보량 증가의 결과인지 완전히 분리하기는 어렵다. 그러나 LCR에서의 차이가 +1.4%p이지만 CDR에서는 +82.5%p의 큰 격차가 나타났다는 점은 정보량보다는 구조화 형식이 주요 요인으로 볼 수 있다. 이에 대한 ALS 핵심 요소별 요인은 아래와 같다.

(1) WHEN-THEN-BECAUSE 분해

규칙의 조건·행동·근거를 명시적으로 분리함으로써 LLM이 규칙의 적용 범위를 파악하는 데 기여한 것으로 추정된다. 특히 CDR에서 BECAUSE 절이 충돌 판단의 근거를 제공하는 역할을 한 것으로 보인다.

(2) Anti-patterns

금지 행위를 명시적으로 제시하여 negative instruction으로 작용한 것으로 추정된다. CDR에서 에이전트가 Anti-pattern 항목을 참조하여 거부 근거를 제시한 사례가 관찰되었다.

(3) Valid/Invalid Examples

few-shot learning의 원리를 활용하여 규칙 적용의 구체적 양상을 학습하는 데 기여한 것으로 보인다.

다만, 본 실험에서는 요소별 ablation study를 수행하지 않았으므로 각 요소의 개별 기여도는 정성적 추론 수준에 그친다. 향후 연구에서는 문서 분량을 통제하거나 구조 요소를 제거한 ablation study를 통해 이러한 효과를 보다 엄밀히 검증할 필요가 있다.

6.2 외적 타당성

본 연구에서는 단일 도메인인 WMS를 대상으로 수행되었기 때문에 결과의 일반화에는 한계가 있다. WMS는 복잡한 비즈니스 규칙과 다중 조건 분기를 포함하는 특성을 가지므로 다른 도메인(예: 의료, 금융, 물류 등)에서도 유사한 효과가 재현되는지에 대한 추가 검증이 필요하다.

또한, 본 실험은 단일 LLM 모델(Claude Sonnet 4.5)과 단일 성능 티어에 기반하여 수행되었다. 상위 모델(예: Claude Opus)의 경우 더 높은 기본 추론 능력을 바탕으로 설계 문서 없이도 높은 로직 준수율(LCR)을 달성할 가능성이 있으며, 이 경우 그룹 간 성능 차이가 축소될 수 있다. 반대로, 소형 모델에서는 ALS와 같은 구조화된 표현이 더 큰 보조 효과를 제공할 가능성이 있다. 더 나아가 GPT-4, Gemini 등 이종 모델 계열에 대한 일반화 가능성 역시 추가 실험을 통해 검증될 필요가 있다.

6.3 구성 타당성

LCR은 체크리스트 기반으로 정의된 지표로서 생성된 코드가 비즈니스 로직을 포함하고 있는지를 정적 분석을 통해 평가한다. 그러나 이러한 방식이 실제 코드 품질을 완전히 대변한다고 보기는 어렵다. 본 연구는 컴파일 가능성, 런타임 정확성, 성능과 같은 실행 기반 지표를 포함하지 않았으며, 이는 빌드 환경 및 의존성 차이로 인한 외생 변수를 배제하기 위한 선택이다. 그럼에도 불구하고 정적 분석만으로는 로직이 의도대로 실행되는지를 완전히 보장할 수 없다는 한계가 존재한다. 향후 연구에서는 자동화된 통합 테스트를 병행하여 런타임 수준의 정확성까지 평가하는 것이 필요하다.

6.4 신뢰성

LLM의 출력은 확률적 특성을 가지며, 특히 에이전트 기반 다중 턴 환경에서는 상호작용 과정에 따라 결과 변동성이 발생할 수 있다. 본 연구에서는 각 그룹별 5회 반복 실행을 통해 이러한 변동성을 일부 확인하였으나, 표본 크기가 제한적이므로 통계적 해석에는 주의가 필요하다.

또한, 본 실험은 Python 스크립트를 통해 CLI를 순차적으로 호출하는 자동화된 방식으로 수행되었다. 이는 인간 개발자가 AI의 출력을 검토하고 추가 지시를 제공하는 실제 대화형 개발 환경과는 차이가 있다. 아울러 rate limit을 회피하기 위해 호출 간 대기 시간을 삽입하였으며, 이는 실험 환경상의 제약 조건일 뿐 방법론적 요소는 아니다.

VII. 결 론

본 연구는 LLM 기반 코드 생성에서 구조화된 비즈니스 로직 표현이 코드 품질에 미치는 영향이라는 문제의식 기반하여, 비즈니스 로직을 마크다운 기반의 원자적 단위로 분해, 조직화한 Atomic Logic Sheet(ALS)와 이를 지원하는 Hierarchical Business Logic Injection(H-BLI) 3계층 프레임워크를 제안하고, AI 코딩 에이전트(Claude Code CLI) 환경에서 그 효과를 실증적으로 분석하였다. WMS 도메인을 대상으로 98개의 LCR 항목, 8개의 CDR 시나리오, 19개의 데이터 테이블로 구성된 복합 규칙 환경에서, 요구사항만 제공한 경우, 자연어 기반 설계 문서를 제공한 경우, ALS를 적용한 경우의 세 그룹 비교 실험을 수행하였다. 그 결과, ALS 기반 구조화 방식은 단순한 정보량 증가 효과를 넘어, LLM의 sycophancy 현상을 완화하고 규칙 해석의 일관성을 유지하는 데 독립적으로 기여함을 확인하였다. 본 연구의 핵심 결론은 다음과 같다.

첫째, 로직 준수율(LCR) 측면에서는 설계 문서의 존재 여부가 중요한 요인으로 작용하는 경향이 관찰되었다. 요구사항만 제공된 경우 대비 자연어 설계 문서를 포함한 경우 +4.0%p의 유의미한 개선이 확인되었으며, ALS 구조화는 추가적으로 +1.4%p의 제한적인 향상을 보였다. 이는 로직 구현에서는 정보의 구조화보다 정보의 충분성이 우선적으로 작용하는 경향을 시사한다.

둘째, 충돌 탐지율(CDR)에서는 전혀 다른 양상이 나타났다. 요구사항만으로는 충돌 탐지가 전혀 이루어지지 않았으며(0.0%), 자연어 설계 문서만으로도 일관된 탐지가 어려웠다(17.5%). 반면 ALS는 규칙 번호 체계와 BECAUSE 절을 기반으로 모든 시나리

오에서 100%의 탐지율을 달성하였다. 이는 충돌 탐지와 같은 고차원적 판단에서는 정보의 “구조화 방식”이 중요한 역할을 하는 것으로 관찰되었다.

셋째, 스키마 일탈 건수(SDC) 분석 결과, 설계 문서의 존재는 스키마 안정성 확보에 있어 필요하다고 판단될 수 있다. 요구사항만을 기반으로 한 경우 대규모 스키마 누락이 발생한 반면, 설계 문서를 포함한 경우 일탈이 95% 이상 감소하였다. 다만 SDC에서는 자연어 설계 문서가 ALS보다 더 안정적인 결과를 보였으며, 이는 특정 실행에서 발생한 이상치의 영향으로 해석된다. 전반적으로 SDC는 구조화 형식보다는 설계 정보의 포함 여부에 더 크게 의존하는 것으로 나타났다.

결과적으로 본 연구는 LLM 기반 개발에서 성능을 결정짓는 요인이 단일하지 않으며, 과업 유형에 따라 정보의 “양”과 “구조”가 서로 다른 중요도를 가진다는 점을 실증적으로 보여준다.

위에서 도출된 핵심 결론으로부터 다음과 같은 세 가지 핵심 시사점을 나타낼 수 있다.

첫째, 설계 문서의 존재는 에이전트 기반 개발 환경에서 코드 품질을 결정짓는 주요 변수가 될 수 있을 것이다. 요구사항만을 제공하는 바이브 코딩 방식은 세부 규칙 누락, 심각한 스키마 이탈, 그리고 충돌 미탐지로 이어졌으며, 에이전트의 반복적 수정 능력만으로는 이를 보완하지 못하였다. 특히 초기 설계 오류가 누적 개발 과정에서 전파되며 품질 저하가 증폭되는 경향이 확인되었다.

둘째, ALS의 차별적 가치는 충돌 탐지에서 두드러진다. 로직 구현 정확도(LCR)에서는 구조화 효과가 제한적이었으나, 충돌 탐지(CDR)에서는 매우 큰 성능 향상이 나타났다. 이는 로직 구현에는 정보의 충분성이 충돌 탐지와 같은 고차원적 판단에는 정보의 구조화가 중요한 역할을 하는 것으로 관찰되었다. 특히 ALS의 규칙 번호 체계와 Logic Anchoring 메커니즘은 에이전트가 규칙을 안정적으로 참조하고 근거 기반으로 응답하도록 유도함으로써 sycophancy를 억제하는 효과적인 수단으로 작용할 가능성을 보였다.

셋째, 설계 정보의 제공은 스키마 안정성 확보에 필수적이다. 설계 문서가 없는 경우 대규모 스키마 일탈이 발생한 반면, 설계 문서를 포함한 경우 이러한 문제가 크게 감소하였다. 다만 스키마 일탈 측면에서는 구조화 방식보다는 설계 정보의 존재 여부가 주요 결정 요인으로 작용하였다.

본 연구의 평가는 생성 코드의 정적 분석에 기반하며 컴파일 가능성 및 런타임 동작 정확성은 검증하지 않았다. 또한 n=5의 탐색적 연구로서, LCR의 C−B 차이(+1.4%p)와 같은 소폭 차이에 대해서는 통계적 유의성을 주장할 수 없다.

본 연구의 결과를 바탕으로, 향후 연구는 다음 네 가지 방향으로 확장될 필요가 있다. 첫째, 반복 실험과 자동화된 검증 체계를 통해 LCR·CDR·SDC 지표의 통계적 유의성을 보다 엄밀히 검증할 필요가 있다. 둘째, WHEN-THEN-BECAUSE 구조, Anti-patterns, Valid/Invalid Examples 등 ALS 구성 요소별 ablation study를 수행하여 각 요소의 기여도를 정량적으로 분석해야 한다. 셋째, WMS와 단일 모델에 한정된 실험 범위를 의료·금융·물류 등 다양한 도메인과 GPT·Gemini 계열 모델로 확장하여 ALS의 일반화 가능성을 검증할 필요가 있다. 넷째, 자연어 기반 요구사항과 설계 문서로부터 H-BLI 구조에 적합한 ALS를 자동 생성하는 변환 파이프라인을 개발함으로써 실무 적용성과 확장성을 확보해야 한다. 이러한 후속 연구를 통해 ALS는 LLM 기반 소프트웨어 공학(AI4SE)에서 활용 가능한 구조화 설계 표준으로 발전할 수 있을 것으로 기대된다.

References

[1] M. Chen, et al., "Evaluating Large Language Models Trained on Code", arXiv preprint, arXiv:2107.03374, pp. 1-35, Jul. 2021. https://doi.org/10.48550/arXiv.2107.03374.

[2] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A Survey on Large Language Models for Code Generation", ACM Trans. Softw. Eng. Methodol. (TOSEM), Vol. 35, No. 2, pp. 1-72, Feb. 2026. https://doi.org/10.1145/3747588.

[3] J. Choi, "ChatGPT-based Software Requirements Engineering", Journal of Internet of Things and Convergence, Vol. 9, No. 6, pp. 45-50, Dec. 2023. https://doi.org/10.20465/KIOTS.2023.9.6.045.

[4] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation", Advances in Neural Information Processing Systems, New Orleans, LA, USA, Vol. 36, pp. 21558-21572, Dec. 2023. https://doi.org/10.48550/arXiv.2305.01210

[5] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Liang, Y. Li, Q. Wang, and T. Xie, "CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models", Proc. ICSE, Lisbon, Portugal, pp. 1-12, Apr. 2024. https://doi.org/10.1145/3597503.3623316

[6] M. Sharma, et al., "Towards Understanding Sycophancy in Language Models", Proc. ICLR, Vienna, Austria, pp. 1-35, May 2024. https://doi.org/10.48550/arXiv.2310.13548.

[7] L. Malmqvist, "Sycophancy in Large Language Models: Causes and Mitigations", Proc. CompCom, Lecture Notes in Networks and Systems, Vol. 1426, pp. 61-74, Aug. 2025. https://doi.org/10.1007/978-3-031-92611-2_5.

[8] F. Liu, Y. Liu, L. Shi, Z. Yang, L. Zhang, X. Lian, Z. Li, and Y. Ma, "Beyond Functional Correctness: Exploring Hallucinations in LLM-Generated Code", arXiv preprint, arXiv:2404.00971, pp. 1-21, Apr. 2024. https://doi.org/10.48550/arXiv.2404.00971

[9] J. Oh, J. Choi, H.-Y. Kim, and H. Song, "Improving Language Model Quality through LLM-based Fine-Grained Hallucinated Summary Generation", KIISE Transactions on Computing Practices, Vol. 31, No. 2, pp. 91-97, 2025년. https://www.kci.go.kr/kciportal/ci/sereArticleSearch/ciSereArtiView.kci?sereArticleSearchBean.artiId=ART003173654.

[10] T. Ridnik, D. Kredo, and I. Friedman, "Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering", arXiv preprint, arXiv:2401.08500, pp. 1-10, Jan. 2024. https://doi.org/10.48550/arXiv.2401.08500

[11] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, C. Wang, S. Liu, and Q. Wang, "ClarifyGPT: A Framework for Enhancing LLM-Based Code Generation via Requirements Clarification", Proc. ACM Softw. Eng., Vol. 1, No. FSE, pp. 2332-2354, Jul. 2024. https://doi.org/10.1145/3660810.

[12] Y. Zi, H. Menon, and A. Guha, "More Than a Score: Probing the Impact of Prompt Specificity on LLM Code Generation", Proc. IJCNLP-AACL, Mumbai, India, pp. 2380-2402, Dec. 2025. https://doi.org/10.18653/v1/2025.ijcnlp-long.128.

[13] R. Koo, M. Lee, V. Raheja, J. I. Park, Z. M. Kim, and D. Kang, "Benchmarking Cognitive Biases in Large Language Models as Evaluators", Findings of ACL, Bangkok, Thailand, pp. 517-545, Aug. 2024. https://doi.org/10.18653/v1/2024.findings-acl.29.

[14] H. Li, X. Tang, J. Zhang, S. Guo, S. Bai, P. Dong, and Y. Yu, "CAUSM: Causally Motivated Sycophancy Mitigation for Large Language Models", Proc. ICLR, Singapore, pp. 1-21, Apr. 2025. https://doi.org/10.48550/arXiv.2410.02506.

[15] H. Jin and H. Chen, "Uncovering Systematic Failures of LLMs in Verifying Code Against Natural Language Specifications", Proc. ASE, Seoul, South Korea, pp. 1-5, Nov. 2025. https://doi.org/10.48550/arXiv.2508.12358.

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", Advances in Neural Information Processing Systems, New Orleans, LA, USA, Vol. 35, pp. 24824-24837, Nov. 2022. https://doi.org/10.48550/arXiv.2201.11903.

[17] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners", Advances in Neural Information Processing Systems, New Orleans, LA, USA, Vol. 35, pp. 22199-22213, Nov. 2022. https://doi.org/10.48550/arXiv.2205.11916.

[18] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-Consistency Improves Chain of Thought Reasoning in Language Models", Proc. ICLR, Kigali Rwanda, pp. 1-24, May 2023. https://doi.org/10.48550/arXiv.2203.11171

저자소개

김 만 수 (Mansu Kim)

2020년 2월 : 국가평생교육진흥원 학점은행제 컴퓨터공학과(공학사)
2025년 2월 : 한국공학대학교 컴퓨터공학과(공학석사)
2025년 3월 ~ 현재 : 한국공학대학교 컴퓨터공학과 박사과정
2025년 2월 ~ 현재 : 화천기계㈜ 정보전략실 부장
관심분야 : AI4SE, LLMOps

최 무 성 (Museong Choi)

2024년 2월 : 한국공학대학교 게임공학과(공학사)
2026년 2월 : 한국공학대학교 컴퓨터공학과(공학석사)
2026년 3월 ~ 현재 : 한국공학대학교 컴퓨터공학과 박사과정
관심분야 : AI 헬스케어, 이상탐지, 딥러닝

왕 은 영 (Eunyoung Wang)

2000년 2월 : 서울과학기술대학교 전자계산학과(공학사)
2021년 2월 : 연세대학교 디자인경영학과(이학석사)
2025년 2월 : 한국공학대학교 IT반도체융합공학과(공학박사)
2025년 2월~2026년 2월 : 숙명여자대학교 SW중심대학 특임교수
관심분야 : HCI, LLM 모델, AI 헬스케어

정 성 택 (Sungtaek Chung)

1992년 2월 : KAIST 전기 및 전자공학과(공학사)
1995년 2월 : KAIST 정보 및 통신공학과(공학석사)
2000년 2월 : KAIST 전기 및 전자공학과(공학박사)
2004년 3월 ~ 현재 : 한국공학대학교 컴퓨터공학과/인공지능학과 교수
관심분야 : AI 의료 신호 및 영상처리, AI 헬스케어