코드 생성을 위한 LLM 환각 완화에 관한 체계적 문헌 예비 연구

김장현°, 조은호, 고인영
전산학부, 한국과학기술원
{big01ad,ehcho,iko}@kaist.ac.kr

A Preliminary Systematic Literature Review on Mitigating LLM Hallucination for Code Generation

Janghyun Gim, Eunho Cho, In-Young Ko
School of Computing, Korea Advanced Institute of Science and Technology(KAIST)

요 약

대규모 언어 모델은 코드 생성 분야에 혁신을 가져왔으나, 사실에 근거하지 않거나 존재하지 않는 요소를 생성하는 환각(hallucination) 현상은 심각한 도전 과제로 남아있다. 이러한 환각은 생성된 코드의 신뢰성과 정확성을 저해해 소프트웨어 개발 수명 주기에 중대한 위험을 초래할 수 있다. 본 연구는 LLM의 코드 생성 환각 문제를 완화하기 위한 소프트웨어 공학 분야의 연구 동향을 체계적으로 분석하고 종합하는 것을 목표로 한다. 이를 위해 Scopus, IEEE Xplore, ACM Digital Library 데이터베이스에서 2020년부터 2025년 9월까지 출판된 문헌을 대상으로 체계적 문헌 예비 연구를 수행했다. PRISMA 2020 지침에 따라 2단계 스크리닝 과정을 거쳐 최종적으로 19편의 핵심 연구를 선정했다. 예비적 조사 결과, 코드 환각 완화 연구는 RAG를 필두로 한 네 가지 주요 완화 패러다임이 연구를 주도하고 있음을 확인했으며, 추후 연구에서는 이 결과를 기반으로 본 SLR을 수행해 더 구체적인 코드 생성 LLM의 환각 현상 완화에 대한 분석을 진행할 예정이다.

1. 서론

대규모 언어 모델(Large Language Model, LLM)을 활용한 코드 생성 분야에서 환각(hallucination) 현상은 핵심적인 도전 과제로 대두되었다[1]. 환각은 LLM이 사실에 근거하지 않은 코드를 생성하는 현상을 지칭하며, 존재하지 않는 API를 호출하거나[2], 유효한 API를 부정확하게 사용하는 오류를 포함한다[3]. 이러한 오류는 생성된 코드의 신뢰성과 정확성을 심각하게 훼손하며, LLM 기반 코드 생성 기술의 실용화를 가로막는 주요 걸림돌이 된다. 따라서 인공지능을 이용한 코드 개발에 있어 이러한 환각 현상을 완화하는 방법에 대한 연구가 시급하다.

이 문제를 해결하기 위해 소프트웨어 공학 분야 연구자들은 LLM 환각 완화를 위한 다양한 기법을 제시해 왔다. 코드 생성 시 LLM에게 공식 API 문서를 검색해 제공하는 검색 증강 생성(Retrieval-Augmented Generation, RAG) 기법[4-5]이나, 생성된 코드의 정확성을 입력 변환으로 생성된 서로 관련된 입력 값으로 프로그램을 여러 번 실행했을 때의 출력 값 사이에 유지되어야 하는 관계인 메타모픽 관계를 활용하는 메타모픽 테스팅(Metamorp-hic Testing)으로 검증하는 방법[6] 등이 제안되었다.

하지만 LLM이 코드 생성 시 일으키는 환각 현상을 완화하기 위한 소프트웨어 공학적 접근법을 종합적으로 정리한 연구는 부재했다. 이에 본 연구는 예비적인 체계적 문헌 조사(Systematic Literature Review, SLR)를 통해 관련 연구 동향을 분석하고, 추후 SLR에서 사용하는 검색어(Query), 포함•배제 기준(Inclusion•Exclusion Criteria) 및 연구 질문(Research Question)를 찾고자 한다.

이 논문은 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(RS-2025-02218761, 50%), (RS-2024-00406245, 25%)과 삼성전자(25%)의 지원을 받아 수행된 연구임.

2. 체계적 문헌 조사 방법론

본 연구는 코드 생성 LLM의 환각 완화에 대한 소프트웨어 공학적 접근법을 식별하고 종합하기 위해 PRISMA 2020[7] 지침을 따라 SLR을 설계했다. 연구의 재현성을 보장하기 위해 모든 과정은 단일 연구자에 의해 수행되었으며, 프로토콜을 사전에 정의해 일관성을 유지했다.

다만 본 연구에 앞서 진행하는 체계적 문헌 예비 연구인 만큼 스노우볼링(Snowballing)[8] 등 일부 절차를 진행하지 않고, 문헌 탐색 등 체계적 문헌 연구의 핵심적인 부분만 우선적으로 진행했다. 이를 통해 추후 진행할 본 체계적 문헌 연구에서 사용할 수 있는 검색어, 포함 기준, 배제 기준 및 연구 질문을 찾고 본 연구에 앞서 연구 결과가 어떻게 나올지 미리 예측하는 것이 목표이다.

2.1 연구 질문 (Research Questions)

본 SLR은 코드 생성 LLM의 환각 완화 연구 지형을 다각도로 이해하기 위해 아래와 같이 총 3개의 연구 질문(RQ)을 설정했다.

RQ1. 정의 및 동향(Definition & Trends): 코드 생성 LLM의 환각은 어떻게 정의되고 있으며, 관련 연구 동향은 어떠한가?

RQ2. 요구, 전략, 적용(Requirements, Strategies & Applicati-ons): LLM 코드 환각을 완화하기 위해 어떤 핵심 전략들이 제안되었으며, 이는 어떤 아키텍처 패턴으로 구현되는가?

RQ3. 평가 (Evaluation): 제안된 완화 전략의 효과는 어떤 지표와 벤치마크를 통해 평가되는가?

2.2 문헌 검색 전략

2025년 9월 5일, 주요 학술 데이터베이스인 Scopus, IEEE Xplore, ACM

Digital Library(DL)에서 문헌 검색을 수행했다. 영어로 작성되어 2020년부터 2025년 내에 출판된 동료 심사(peer-reviewed) 논문으로 한정했다.

검색어는 'LLM', 'hallucination', 'code generation', 'mitigation'과 관련된 핵심 키워드와 동의어를 조합해 구성했으며, 각 데이터베이스의 검색 문법에 맞춰 조정되었다. 실제 검색어는 [9]에서 확인할 수 있다. 검색어는 4개로 분해해 정밀도와 재현성을 확보하도록 설계했다.

A. **현상(Phenomenon)**: hallucination의 동의어(예: "factual error*", fabricat, invent 등)를 포함해 표현 다양성을 포괄.

B. **모델 범위(Model Scope)**: "large language model*", LLM, "code LLM", 등으로 대상 모델군을 명시.

C. **SE 맥락(SE Context)**: "code generation", "source code" 등으로 소프트웨어 공학 산출물을 요구.

D. **완화 접근(Mitigation Methods)**: mitigat*, verification, "self-check" 등으로 실제 완화 행위/메커니즘을 포괄.

각 데이터베이스의 필드/문법 제약에 맞추어 A-D 블록을 AND 결합하되, IEEE Xplore/ACM DL에서는 Title/Abstract 중심 필드 제한, Scopus에서는 SUBJAREA(컴퓨터공학/공학)와 *DOCTYPE(회의·저널) 등으로, 관련 없는 논문들을 구조적으로 억제했다. 2020년 하한은 코드 LLM/도구증강 기법의 본격화 시기 이후의 문헌을 포착하려는 의도에서 설정했다.

2.3 문헌 선정 기준

수집된 문헌은 2단계에 걸친 스크리닝 과정을 통해 최종 분석 대상을 선정했다. 1단계에서는 제목과 초록을 검토했고, 2단계에서는 전문을 검토했다. 각 단계에서는 아래의 포함 기준과 배제 기준을 엄격하게 적용했다.

포함 기준 (Inclusion Criteria)
(IC1) 환각 완화 기법을 명시적으로 제안하거나 평가하는 연구
(IC2) 코드, 테스트 등 소프트웨어 공학 산출물을 다루는 연구
(IC3) RQ 중 하나 이상에 답할 수 있는 경험적 증거를 포함한 연구
(IC4) 영어로 작성된 동료 심사 출판물
(IC5) 데이터 추출 및 재현이 가능할 만큼 상세히 보고된 연구
(IC6) 정량적 완화 지표 및 베이스라인 비교를 포함한 연구
배제 기준 (Exclusion Criteria)
(EC1) 소프트웨어 공학 분야와 직접적인 관련이 없는 연구
(EC2) 2차 연구(SLR/SMS), 학위 논문, 비심사 출판물 등
(EC3) 구체적 완화 기법 평가가 없는 연구
(EC4) AI 모델과 관련 없는 전통적인 소프트웨어 공학 연구
(EC5) 영어가 아닌 언어로 작성된 연구
(EC6) 환각이 아닌 정확성, 생산성 등에만 초점을 맞춘 연구

본 포함/배제 기준은 연구 범위 적합성, 연구의 방법론적 엄밀성, 재현성 및 비교가능성을 확보하기 위해 설계했다. 특히 본 SLR은 LLM 기반 코드 생성 맥락의 환각 완화를 핵심 구성개념으로 정의하고, 소프트웨어 공학적 산출물과 정량적 평가를 동반한 1차 연구에 한정함으로써, 후속 분석 및 실증적 비교가 가능하도록 했다.

3. 체계적 문헌 조사 결과

3.1. 문헌 선정 결과

그림 1은 문헌 선정 결과를 PRISMA 도식도로 보여준다. 초기 검색 및 수동 추가를 통해 총 387편의 문헌을 식별했다. 이 중 69편의 중복 문헌을 제거한 후, 제목과 초록을 기반으로 한 1단계 스크리닝을 통해 288편을 배제해 총 357편의 논문을 배제했다. 이후 총 30편의 논문이 후보로 남았으며, 전문 검토 기반의 2단계 스크리닝을 진행했다. 최종적으로 11편의 논문이 2단계에서 추가로 배제되어, 총 [1-6, 11-22]에 해당하는 19편의 논문이 본 연구의 분석 대상으로 선정되었다.

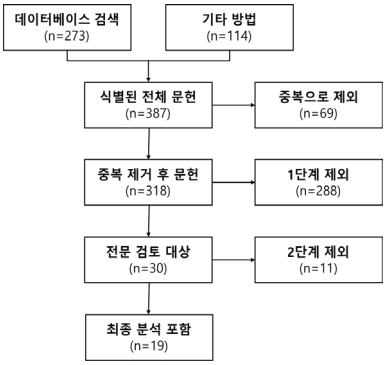

그림 1. PRISMA 도식도로 표현한 예비적 SLR 논문 선정 결과

3.2 연구 질문에 대한 답변

3.2.1 (RQ1) 정의 및 동향

코드 생성 AI에서 환각은 구문적으로는 그럴듯하지만 주어진 맥락, 프로그래밍 언어 표준, 또는 라이브러리 문서에 비추어 볼 때 의미론적으로나 사실적으로 부정확한 코드 또는 관련 산출물을 생성하는 현상으로 정의된다[1]. 이는 단순한 문법 오류를 넘어, 컴파일러가 잡아내지 못하는 교묘한 형태의 사실적 불일치를 포함한다. 이러한 환각은 4가지로 분류된다.

- **API 및 라이브러리 환각**: 존재하지 않는 패키지[10]·메서드를 사용하거나, 구식(deprecated) API를 참조하는 등[2] 라이브러리를 부정확하게 호출하는 경우이다. 가장 빈번하게 다뤄지는 유형이다[11-12].
- **구조적 및 구문적 환각**: 프로그래밍 언어의 근본적인 문법 규칙을 위반하는 코드를 생성한다. 문법이 엄격한 하드웨어 기술 언어(HDL) 분야에서 중요한 문제로 다뤄진다[13-15].
- **논리적 환각**: 코드가 오류 없이 실행되지만 사용자의 의도와 다른 결과를 출력하거나 알고리즘에 결함이 있는 경우이다. 탐지가 가장 어려우며, C/C++ 코드 리펙토링[16] 및 복구[17] 과정에서 나타난다.
- **귀속 환각**: 코드의 실제 기능과 주석 또는 기술 보고서[18], RFC 문서[19]의 설명이 일치하지 않는 경우이다.

이러한 환각 완화에 대한 연구는 매우 최근에 시작되어 폭발적으로 증가하는 추세를 보인다. 분석 대상이 된 19개 주요 연구의 대다수는 2024년과 2025년에 출판되었거나 출판될 예정으로, 연구 활동이 극히 최근에 집중되어 있다. 이는 코드 생성 AI에 대한 연구 초점이 LLM의 코딩 능력 입증 단계에서 생성된 코드의 신뢰성 확보 단계로 전환되고 있음을 시사한다.

또한 이 분야의 연구는 **실증적 실험(Empirical Experimentation)** 방법론이 압도적인 주류를 이룬다. 분석된 19개 연구 모두가 실험 범주에 해당하며, 일반적으로 다음 3단계 절차를 따른다.

1. 새로운 완화 기법(예: RAG, 새로운 프롬프팅 전략)을 제안.
2. 환각이 발생하기 쉬운 작업이 포함된 벤치마크를 구축•채택.
3. 제안 기법의 성능(예: 정확도, 테스트 통과율)을 베이스라인 (완화 전략이 없는 순수 LLM 등)과 정량적으로 비교 평가.

3.2.2 (RQ2) 요구, 전략, 적용

문헌에서 제안된 핵심 완화 전략들은 LLM의 생성 프로세스에 개입하는 시점에 따라 명확한 아키텍처 패턴으로 구현된다. 주요 전략과 패턴은 다음과 같이 분류할 수 있다.

1. **생성 전(Pre-Generation) 패턴**: 검색을 통한 컨텍스트 강화
2. **생성 후(Post-Generation) 패턴**: 실행을 통한 검증 및 개선
3. **생성 중(During Generation) 패턴**: 디코딩 제약
4. **데이터/모델 중심 패턴**: 도메인 지식 내재화

가장 지배적인 아키텍처 패턴은 검색기-생성기(Retriever-Gener-ator) 구조를 사용하는 것이다. 검색 증강 생성을 통해 코드 생성에 필요한 외부

최신 문서(예: API 문서, 기술 블로그)를 먼저 검색하고, 이를 LLM의 입력 컨텍스트로 함께 제공한다[4-5]. 이는 LLM이 부정확한 내재 지식에 의존하는 대신, 사실에 근거한 코드 생성을 유도하는 보편적인 접근법이다.

두 번째 주요 패턴은 생성-검증-개선(Generate-Verify-Refine)이라는 피드백 루프를 구축하는 것이다. 반복적 개선(Iterative Refinement)을 활용해, LLM이 생성한 코드를 즉시 컴파일하거나 테스트 케이스를 실행해 검증한다. 만약 오류가 발생하면, 그 오류 메시지를 피드백으로 삼아 LLM에게 코드 수정을 요청하는 사이클을 반복한다[20-21]. 이는 코드 분야의 고유한 장점인 소프트웨어의 결과를 검증할 수 있는 실행 가능한 '테스트 오라클(Test Oracle)'을 활용하는 강력한 전략이다.

상대적으로 드물지만, 생성 과정을 제어하는 패턴도 존재한다. 제약된 디코딩(Constrained Decoding)은 생성되는 토큰(Token)이 프로그래밍 언어의 문법 규칙을 위반하지 않도록 강제해 구조, 구문적 환각을 원천적으로 방지할 수 있다[2].

위 패턴들과 달리, 추론 시점이 아닌 모델 자체를 변경하는 접근법도 존재한다. 도메인 특화 미세조정(Domain-Specific Fine-tuning)은 특정 도메인(예: 하드웨어 설계)의 고품질 데이터셋으로 LLM을 추가 학습시켜, 해당 분야의 규칙과 지식을 모델에 내재화하는 방식이다[11-12]. 이는 특정 분야에서 최고의 성능을 보이지만, 비용과 노력이 많이 든다.

3.2.3 (RQ3) 평가

제안된 완화 전략의 효과는 주로 특정 유형의 환각을 정량적으로 측정하기 위해 설계된 특화 벤치마크와 세분화된 지표를 통해 평가된다. 하지만 현재의 평가 방식은 현실성이 부족하다는 명확한 한계를 동시에 보여주고 있다. 연구자들은 완화 기법의 성능을 객관적으로 비교하기 위해 다음과 같은 벤치마크와 지표들을 개발해 사용하고 있다.

- **APIHulBench**[2]: API 호출 관련 환각(존재하지 않는 함수 호출 등)을 평가하기 위해 특별히 설계된 벤치마크
- **패키지 환각 데이터셋**[10]: LLM이 추천하는 패키지 이름이 실제 패키지 저장소에 존재하는지 교차 검증해, 소프트웨어 공급망 공격으로 이어질 수 있는 보안 관련 환각을 평가
- **HDL 특화 벤치마크**[13]: 하드웨어 기술 언어(HDL)의 구문적 정확성 및 물리적 칩으로의 합성 가능성(synthesizability)과 같이 엄격한 기준을 테스트하기 위한 문제들로 구성
- **실행 정확성 지표**[1, 4]: 생성된 코드가 주어진 테스트를 통과하는 비율(Pass@k), 컴파일 성공 여부 등 기능적 정확성 평가
- **API 특화 지표**[2, 11-12]: 잘못된 API를 사용한 비율이나, 더 이상 사용되지 않는 구식(deprecated) API를 사용한 비율(DUR)을 측정해 코드의 유지보수성과 품질을 평가
- **사실성 및 일관성 지표**[6]: 생성된 설명이 코드의 실제 기능과 일치하는지(지식 F1), 코드가 논리적 제약(메타모픽 관계)을 만족하는지 평가

4. 논의

본 예비적 SLR을 통해 현재까지 코드 생성 AI가 겪고 있는 환각이 어떻게 정의되고, 소프트웨어 공학적인 접근으로 이를 완화시키는 전략이 어떤 아키텍처와 패턴을 가지고 있는지 알 수 있었다. 그리고 완화 전략이 어떤 벤치마크와 지표를 사용해서 평가하는 지 알 수 있었다. 이 결과를 토대로 본 SLR을 위한 연구 질문과 검색어와 포함/배제 기준이 현 연구에서 놓치고 있는 점을 찾을 수 있는 등 적합함을 알 수 있었다.

현재까지의 연구에서는 현 아키텍처 내에서 코드 생성 이후 개발자의 코드에 대한 제안 수락, 거부, 수정 등 개발자의 피드백을 반영하지 못했다. 실제 사용 환경에서 개발자의 상호작용을 피드백으로 활용해 모델을 지속적으로 개선하는 모니터링 시스템에 대한 연구를 통해 이를 개선할 수 있을 것으로 기대한다. 벤치마크의 측면에서 분석할 때 API 환각, 패키지 환각 등 특정 환각에 집중해 나온 만큼 이를 포함해 환각에 대한 표준 벤치마크를 구축하는 연구 또한 필요하다. 마지막으로 현 연구 대다수에서는 코드 내 함수 단위에서의 환각 완화에 집중하고 실제 코드 리포지토리/전역 환경에 대한 연구는 소수에 그친다[1-2]. 실제 개발에서는 여러 파일에 걸쳐 개발을 하는 만큼 의존성을 고려해 코드 리포지토리에 대한 완화 기법 또는 이를 확인할 수 있는 벤치마크에 대한 추가 연구가 더 필요하다.

본 SLR에 앞서 연구 질문과 포함/배제 기준 등 예비 연구의 부족한 점을 개선할 필요가 있다. 연구 질문의 경우, 코드 생성 AI의 환각 완화에 대한 현재까지의 연구 동향을 잘 파악할 수 있었으나 현재 연구들이 가진 연구 공백과 한계점을 찾는 데는 어려움이 있었다. 이를 개선하기 위해 현 연구들의 한계점을 찾을 수 있는 질문을 추가할 필요가 있다. 마지막으로 위 연구의 문헌 선정 과정이 단일 연구자에 의해 수행되는 등 편향의 가능성이 존재하는 과정을 수정할 필요가 있다.

5. 결론

본 연구는 코드 생성 LLM의 환각 현상을 완화하기 위한 소프트웨어 공학적 접근을 필두로 한 연구 동향을 파악하고자 체계적 문헌 예비 연구를 수행했다. 이 과정을 통해 관련 연구를 식별하기 위한 명확한 포함/제외 기준과 분석의 틀이 되는 연구 질문을 수립하고, 19개의 핵심 주요 연구를 선정해 분석했다. 예비적 조사 결과, 코드 환각 완화 연구는 2024-2025년에 폭발적으로 증가하는 최신 분야이며, RAG를 필두로 한 네 가지 주요 완화 패러다임이 연구를 주도하고 있음을 확인했다. 또한 개발자의 피드백을 기반으로 한 모니터링의 필요성과 환각에 대한 표준 벤치마크를 구축하는 연구의 필요성을 확인했다. 그리고 SLR을 위한 연구 질문과 검색어와 포함/배제 기준이 현 연구에서 놓치고 있는 점을 찾는 등 일정 부분 적합함을 알 수 있었다. 이후 이러한 결과를 기반으로 본 SLR을 수행해 더 구체적인 코드 생성 LLM의 환각 현상 완화에 대한 분석을 진행할 예정이다.

참고문헌

[1] Z. Zhang, C. Wang, Y. Wang, E. Shi, Y. Ma, W. Zhong, J. Chen, M. Mao, and Z. Zheng, "LLM Hallucinations in Practical Code Generation: Phenomena, Mechanism, and Mitigation," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, Art. no. ISSTA022, Jul. 2025.

[2] Y. Chen, M. Chen, C. Gao, Z. Jiang, Z. Li, and Y. Ma, "Towards Mitigating API Hallucination in Code Generated by LLMs with Hierarchical Dependency Aware," in *Proc. 33rd ACM Int. Conf. Found. Softw. Eng.* (FSE) - Companion, Trondheim, Norway, 2025.

[3] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: large language model connected with massive APIs," in *Proc. 38th Int. Conf. Neural Inf. Process. Syst.* (NeurIPS), Vancouver, BC, Canada, 2024.

[4] N. Jain, R. Kwiatkowski, B. Ray, M. K. Ramanathan, and V. Kumar, "On Mitigating Code-LLM Hallucinations with API Documentation," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.: Softw. Eng. Pract.* (ICSE-SEIP), 2025.

[5] Ayala, O., and Bechard, P., "Reducing hallucination in structured outputs via Retrieval-Augmented Generation," in *Proc. 2024 Conf. North Amer. Chapter Assoc. Comput. Linguist.: Hum. Lang. Technol.* (Industry Track), Mexico City, Mexico, Jun. 2024.

[6] W. Wu, Y. Cao, N. Yi, R. Ou, and Z. Zheng, "Detecting and Reducing the Factual Hallucinations of Large Language Models with Metamorphic Testing," *Proc. ACM Softw. Eng.*, vol. 2, no. FSE, Art. no. FSE065, June. 2025.

[7] M. J. Page et al., "The PRISMA 2020 statement: an updated guideline for reporting systematic reviews," *BMJ*, vol. 372, Art. no. n71, Mar. 2021.

[8] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assess. Softw. Eng.* (EASE '14), May 13-14, 2014, Art. no. 38.

[9] drive.google.com/file/d/1ycUFsA0Coyvpg67-KeyYR1UQvfMHPnV.

[10] J. Spracklen, R. Wijewickrama, A. H. M. N. Sakib, A. Maiti, B. Viswanath, and M. Jadliwala, "We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs," in *Proc. USENIX Security Symp.*, 2025.

[11] H. Wei, X. Su, W. Zheng, W. Tao, H. Yu, and Y. Kuang, "Knowledge-guided large language models are trustworthy API recommenders," *Autom. Softw. Eng.*, vol. 32, no. 2, Art. no. 47, 2025.

[12] J. Liu, Y. Zhang, D. Wang, Y. Li, and W. Dong, "THINK: Tackling API Hallucinations in LLMs via Injecting Knowledge," in *Proc. IEEE Int. Conf. Softw. Anal., Evol., Reeng.* (SANER), 2025.

[13] Y. Yang, F. Teng, P. Liu, M. Qi, C. Lv, J. Li, X. Zhang, and Z. He, "HAVEN: Hallucination-Mitigated LLM for Verilog Code Generation Aligned with HDL Engineers," in *Proc. Des., Autom. Test Eur. Conf.* (DATE), 2025.

[14] H. Ping, S. Li, P. Zhang, A. Cheng, S. Duan, N. Kanakaris, X. Xiao, W. Yang, S. Nazarian, A. Irimia, and P. Bogdan, "HDLCoRe: A Training-Free Framework for Mitigating Hallucinations in LLM-Generated HDL," in *Proc. IEEE Int. Conf. LLM-Aided Des.* (ICLAD), 2025.

[15] W. Sun, B. Li, G. L. Zhang, X. Yin, C. Zhuo, and U. Schlichtmann, "Paradigm-Based Automatic HDL Code Generation Using LLMs," in *Proc. Int. Symp. Quality Electron. Des.* (ISQED), 2025.

[16] K. Xu, G. L. Zhang, X. Yin, C. Zhuo, U. Schlichtmann, and B. Li, "HLSRewriter: Efficient Refactoring and Optimization of C/C++ Code with LLMs for High-Level Synthesis," *ACM Trans. Des. Autom. Electron. Syst.*, 2025.

[17] K. Xu, G. L. Zhang, X. Yin, C. Zhuo, U. Schlichtmann, and B. Li, "Automated C/C++ Program Repair for High-Level Synthesis via Large Language Models," in *Proc. 2024 ACM/IEEE Int. Symp. Mach. Learn. for CAD* (MLCAD), 2024

[18] S. B. Tamanna, G. Uddin, S. Wang, L. Xia, and L. Zhang, "ChatGPT Inaccuracy Mitigation during Technical Report Understanding: Are We There Yet?," in *Proc. IEEE/ACM Int. Conf. Softw. Eng.* (ICSE), 2025.

[19] M. Zheng, D. Xie, Q. Shi, C. Wang, and X. Zhang, "Validating Network Protocol Parsers with Traceable RFC Document Interpretation," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, 2025.

[20] J. Liu, J. Yan, Y. Xie, J. Yan, and J. Zhang, "Fix the Tests: Augmenting LLMs to Repair Test Cases with Static Collector and Neural Reranker," in *Proc. 35th IEEE Int. Symp. Softw. Reliab. Eng.* (ISSRE), 2024.

[21] S. K. Ngassom, A. M. Dakhel, F. Tambon, and F. Khomh, "Chain of Targeted Verification Questions to Improve the Reliability of Code Generated by LLMs," in *Proc. ACM Int. Conf. AI-Powered Softw.* (AIware), 2024.

[22] S. Dutta, S. Mahinder, R. Anantha, and B. Bandyopadhyay, "Applying RLAIF for Code Generation with API-usage in Lightweight LLMs," in *Proc. Workshop Nat. Lang. Reasoning and Struct. Expl.* (NURSE@ACL), 2024.