코드 거대 언어 모델 연구 동향

ㅁ **최윤석** / **성균관대학교**

요 약

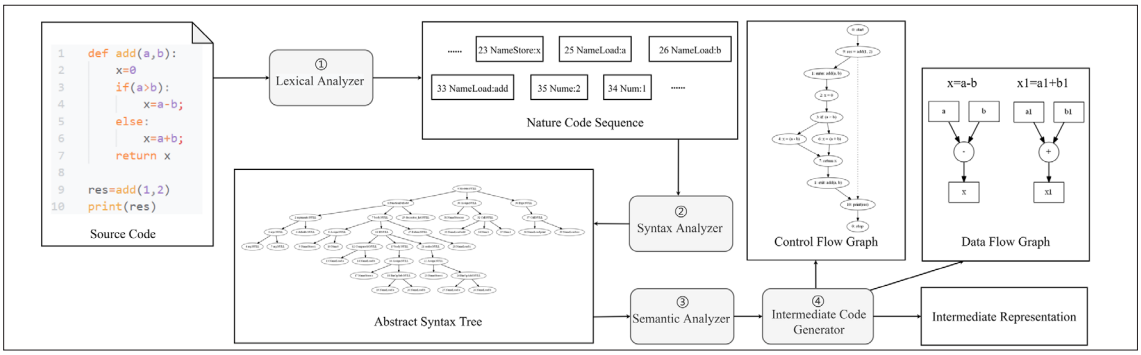
코드 거대 언어 모델(Code Large Language Model, Code LLM)은 자연어 처리 기술을 프로그래밍 언어 영역으로 확장하여, 코드의 이해와 생성, 번역, 검증 등 다양한 코드 인텔리전스(Code Intelligence) 작업을 수행할 수 있는 기술로 주목받고 있다. 본 기고문에서는 코드 LLM의 발전 흐름과 함께, 주요 연구 과제 및 최신 모델의 특징을 살펴본다. 또한 의미 격차(Semantic Gap), 강건성(Robustness), 환각(Hallucination), 평가(Evaluation) 등 현존 모델의 한계와 해결 방향을 논의하고, 향후 연구가 나아가야 할 방향을 제시한다.

I. 서 론

최근 인공지능 분야에서 대규모 언어 모델(Large Language Model, LLM)의 급격한 발전은 자연어 처리(Natural Language Processing, NLP) 영역의 성능을 획기적으로 향상시켰을 뿐 아니라, 그 적용 범위를 소프트웨어 공학과 프로그래밍 언어 처리 영역으로까지 확장시키고 있다. 특히 ChatGPT[1], Claude[2], Gemini[3] 등 범용 LLM의 성공은 언어 이해와 생성 능력이 인간 수준에 근접할 수 있음을 보여주었고, 이는 곧 "언어로 프로그래밍을 이해하고 코드를 작성하는 인공지능"이라는 새로운 연구 패러다임이 등장하였다.

소스 코드는 인간의 언어처럼 일련의 토큰(token)으로 구성된 순차적 데이터이지만, 그 내면에는 인간 언어와는 근본적으로 다른 특성이 존재한다. 프로그래밍 언어는 명령형·함수형·논리형 등 다양한 패러다임을 기반으로 하며, 모든 문장이 정확한 구문(syntax)과 명확한 의미론(semantics)을 충족해야 한다. 자연어가 일정 수준의 모호성과 중의성을 허용하는 반면, 코드는 단 하나의 잘못된 토큰만으로도 전체 프로그램의 실행이 불가능해진다.

또한 코드의 의미는 개별 문장이나 함수 수준을 넘어, 〈그림 1〉과 같이 제어 흐름(Control Flow)이나 데이터 흐름(Data Flow), 그리고 변수나 함수 간의 의존성 관계(Dependency Graph) 등 복잡한 구조적 맥락 속에서 정의된다. 이러한 다층적 구조 때문에 코드 이해는 단순히 단어의 빈도나 문장 구조를 학습하는 언어 처리 문제를 넘어, 논리적 추론(logical reasoning)과 구조적 표현



<그림 1> 소스 코드의 다양한 구조적 표현[4]

(structural representation)을 통합적으로 다루어야 하는 고차원적 인공지능 문제로 확장된다.

이러한 맥락에서 최근 등장한 코드 특화 거대 언어 모델(Code LLM)[5]은 자연어와 프로그래밍 언어 간의 의미적 대응 관계를 학습하여, 코드 이해(understanding), 생성(generation), 요약(summarization), 번역(translation), 취약점 탐지(vulnerability detection) 등 광범위한 코드 인텔리전스(Code Intelligence) 작업을 수행한다. 특히, GitHub, Stack Overflow 등에서 수집한 방대한 규모의 코드-주석(comment) 병렬 데이터는 모델이 코드의 문법적 패턴뿐 아니라, 그 기능적 의도까지 학습할 수 있도록 지원한다[6].

코드 인텔리전스 연구의 궁극적인 목표는 단순히 코드를 "예측"하거나 "완성"하는 것을 넘어서, 프로그래밍 언어를 이해하는 지능적 에이전트(Intelligent Agent)를 구현하는 데 있다. 이러한 에이전트는 코드의 의미를 추론하고, 새로운 요구사항에 따라 적절한 프로그램을 생성하며, 오류를 진단하거나 성능을 최적화할 수 있는 수준의 추론 능력을 지닌다.

최근 각광받고 있는 코드 거대 언어 모델 연구를 중심으로, (1) 코드 인텔리전스의 주요 과제, (2) 사전학습 기반 코드 언어 모델의 발전 양상, (3) 현존 모델의 한계와 도전 과제, (4) 향후 연구 방향을 체계적으로 고찰한다.

II. 코드 인텔리전스의 주요 연구 과제

코드 인텔리전스(Code Intelligence)는 프로그래밍 언어를 이해하고 생성하는 인공지능 기술을 연구하는 분야로, 크게 코드 이해(understanding)와 코드 생성(generation)으로 구분된다. 각 과제는 서로 밀접히 연관되어 있으며, 코드 거대 언어 모델(Code LLM)의 성능과 활용 범위를 결정짓는 핵심 요소로 작용한다.

1. 코드 이해 작업(Code Understanding Task)

코드 이해는 프로그램의 구조적 의미를 파악하고, 코드 간 관계를 식별하는 것을 목표로 한다. 대표적 과제로는 코드 검색(Code Search), 코드 클론 탐지(Code Clone Detection), 취약점 탐지(Vulnerability Detection), 타입 추론(Type Inference) 등이 있다.

코드 검색은 자연어 질의에 부합하는 코드 스니펫을 찾아내는 문제로, 최근에는 자연어와 코드를 동일한 잠재 공간(latent space)에 매핑하여 의미 기반 검색을 수행하는 이중 인코더(Bi-Encoder) 방식이 주류를 이룬다. 코드 클론 탐지는 서로 다른 코드가 동일한 기능을 수행하는지를 식별하는 문제로, 단순한 문법 비교를 넘어 데이터 흐름(Data Flow) 등 구조적 표현을 학습하는 그래프 신경망

(GNN) 기반 접근이 활용된다. 또한 LLM은 코드의 실행 맥락을 이해하여 잠재적 버그를 탐지하거나 변수 타입을 추정하는 등, 전통적 정적 분석을 대체하는 방향으로 발전하고 있다.

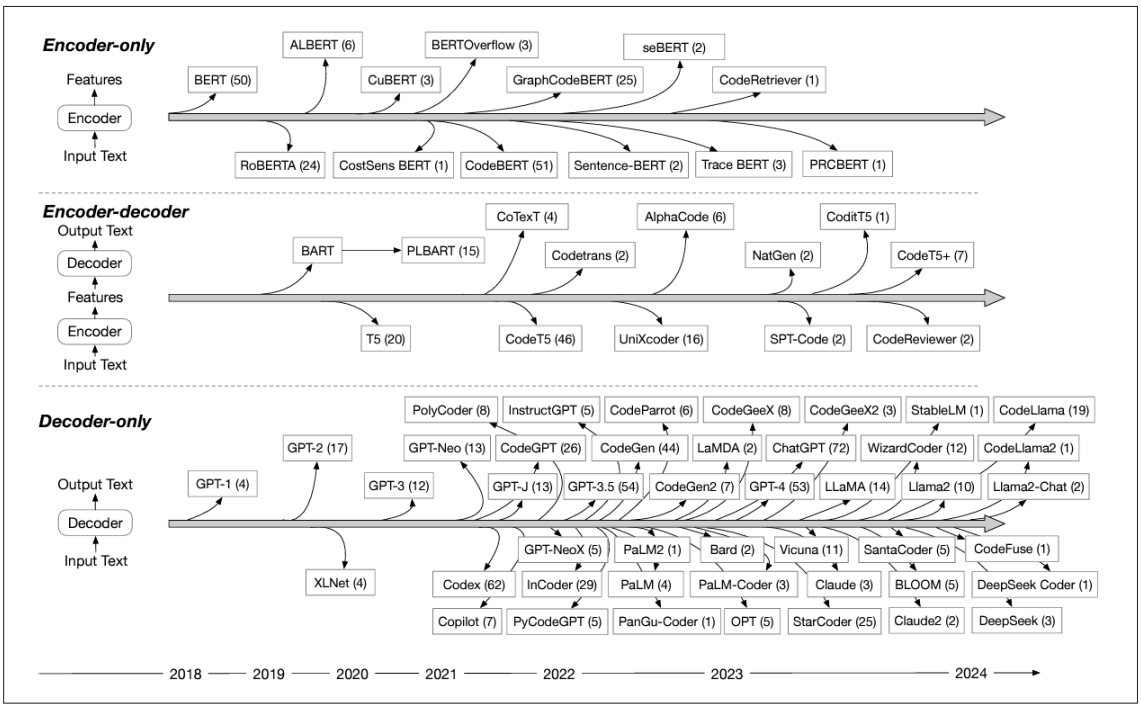
2. 코드 생성 작업(Code Generation Task)

코드 생성은 주어진 요구사항을 만족하는 프로그램을 자동 작성하거나, 기존 코드를 요약 및 번역하는 과제이다. 주요 연구로는 코드 요약(Code Summarization), 코드 번역(Code Translation), 오류 수정(Error Repair), 테스트케이스 생성(Test-case Generation) 등이 있으며, 대부분 자연어-코드 상호작용(NL↔PL)을 기반으로 수행된다.

코드 요약은 함수나 클래스의 동작을 자연어로 설명하는 주석을 자동 생성하는 작업으로, 코드의 구조와 의도를 함께 이해해야 한다. 코드 번역은 언어 간 문법·API 차이를 고려해 기능을 동일하게 유지하는 것이 핵심이다. 오류 수정과 테스트케이스 생성은 코드 실행 결과를 활용해 모델이 스스로 수정하거나 검증하는 실행 피드백 기반(self-refinement) 접근으로 확장되고 있다. 이러한 연구들은 LLM을 단순한 자동화 도구가 아닌 개발자 보조 지능(Assistive Intelligence)으로 발전시키고 있으며, 코드 품질·보안·생산성 측면에서 실제 산업적 효과를 입증하고 있다.

III. 코드 언어 모델의 발전 흐름

코드 언어 모델(Code LLM)의 발전은 자연어 처리에서 검증된 사전학습(Pre-training) 패러다임을 프로그래밍 언어로 확장한 결과라 할 수 있다. 초기에는 언어 모



<그림 2> 코드 특화 거대 언어 모델 동향[7]

델의 구조를 코드 데이터에 단순히 적용하는 수준이었으나, 점차 코드의 문법적 구조(syntactic structure)와 의미적 의존 관계(semantic dependency)를 내재화하는 방향으로 진화하였다. 현재는 수십억~수천억 파라미터 규모의 초대형 모델이 등장하면서, 코드 생성·이해·수정·테스트 등 소프트웨어 개발 전반을 지원하는 단계에 도달하고 있다.

1. 코드 이해 중심 모델

초기의 코드 언어 모델은 BERT 구조를 기반으로 한 양방향 사전학습 모델이 주류를 이루었다. 대표적으로 CodeBERT[5]는 자연어와 프로그래밍 언어의 병렬 데이터를 활용해 코드 이해 및 검색 태스크에서 뛰어난 성능을 보였다. 이 모델은 코드 주석(comment)과 실제 코드 간의 연관성을 학습함으로써, 코드의 기능적 의미를 보다 정교하게 포착할 수 있었다. 이어 GraphCodeBERT[8]는 코드의 데이터 흐름(Data Flow)과 문법적 관계를 그래프 형태로 모델링하여 구문적·의미적 정보를 함께 반영하였다. 이는 코드가 단순한 문장 시퀀스가 아닌 구조적 표현(Structured Representation)을 지닌 언어라는 점을 인식한 첫 시도라 할 수 있다. 이러한 모델들은 코드 검색, 클론 탐지, 타입 추론 등 이해 중심 과제의 기반이 되었다.

2. 코드 생성 중심 모델

이후 연구는 코드 생성 및 번역에 특화된 인코더-디코더 구조(Encoder-Decoder Architecture)로 확장되었다. PLBART[9]와 CodeT5[10]는 코드 생성, 요약, 번역 등 다양한 태스크를 하나의 통합 구조 내에서 수행할 수 있도록 설계되었다. 이들은 코드와 자연어를 동시에 입력받아 의미적 매핑을 학습함으로써, NL↔PL 양방향 변환 능력을 확보하였다. 또한 UniXcoder[11]는 텍스트와 코드를 함께 이해하는 교차 표현(Cross-modal Representation) 학습을 도입하여, 코드 주석 생성과 코드 번역의 성능을 한층 향상시켰다. 최근에는 CodeT5+[12]가 2B~16B 파라미터 규모의 대형 모델로 확장되어, 코드 생성의 품질과 실행 가능성 측면에서 기존 모델을 크게 능가하였다. 이러한 생성 중심 모델의 발전은 LLM이 단순히 코드를 예측하는 수준을 넘어, 코드의 의미적 일관성과 실행 가능한 논리 구조를 함께 학습할 수 있음을 보여준다.

3. 초대규모 공개 모델의 등장

최근에는 초대형 오픈소스 모델들이 등장하며 코드 LLM 연구가 새로운 전환점을 맞이했다.

Code Llama[13]는 Llama 2 아키텍처를 기반으로 한 7B~70B 파라미터 모델로, 코드 이해와 생성 모두에서 뛰어난 범용성을 보였다. CodeGemma[14]는 Gemma 아키텍처를 바탕으로 개발된 경량 모델로, 코드 생성 속도를 약 2배 향상시켜 실시간 응용 환경에 적합한 효율성을 확보하였다. 또한 DeepSeek-Coder V2[15]는 338개 프로그래밍 언어를 지원하며, 최대 128k 토큰의 장문 문맥을 학습할 수 있는 구조를 통해 리포지토리 단위의 코드 이해를 실현하였다.

이러한 모델들은 단순히 매개변수 수를 늘리는 데 그치지 않고, 멀티언어 지원, 대규모 코드 저장소 학습, 문맥적 일관성 유지 등 구조적 개선을 통해 코드 이해의 범위와 정밀도를 크게 확장하고 있다.

IV. 코드 LLM의 주요 기술적 도전 과제

코드 거대 언어 모델은 다양한 프로그래밍 언어를 처리하고 실행 가능한 코드를 생성할 수 있을 만큼 발전하였지만, 여전히 해결해야 할 근본적인 한계와 연구 과제가 존재한다. 본 장에서는 최근 논문과 산업적 논의에서 빈번히 제기되는 네 가지 핵심 도전 과제를 중심으로 살펴본다.

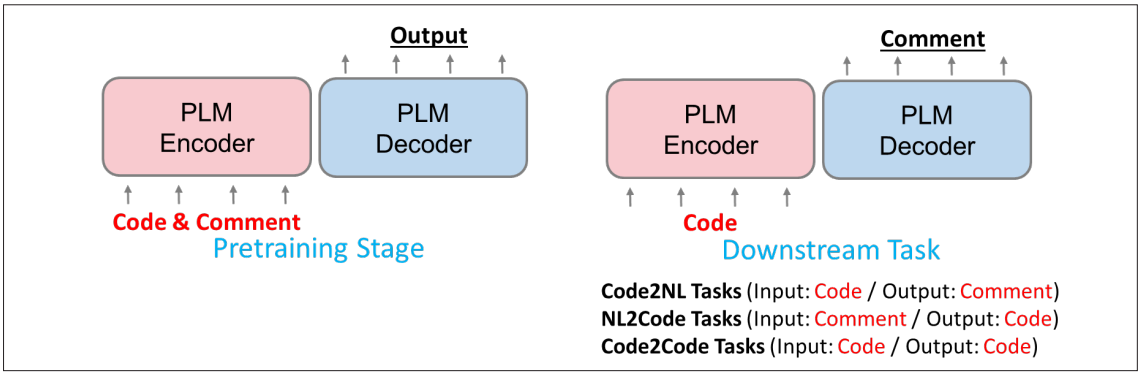
1. 의미 격차(Semantic Gap)

자연어와 프로그래밍 언어 사이의 의미적 불일치(Semantic Gap)는 코드 LLM이 직면한 가장 근본적인 문제 중 하나이다. 대부분의 모델은 사전학습 단계에서 코드와 주석(comment)의 쌍을 학습하지만, 실제 응용 환경에서는 코드만 입력되거나 주석만 존재하는 경우가 많다. 이로 인해 모델은 자연어-코드 간 의미 매핑(NL↔PL alignment)을 완전하게 학습하지 못하고, 특정 태스크에서는 일반화 성능이 크게 저하된다. 이 문제를 해결하기 위한 대표적 시도가 Prefix Tuning 기반의 CodePrompt[16]로, 프롬프트 구조 내에서 태스크별 의미 차이를 줄이기 위해 코드-언어 전이 표현(joint latent representation)을 학습한다.

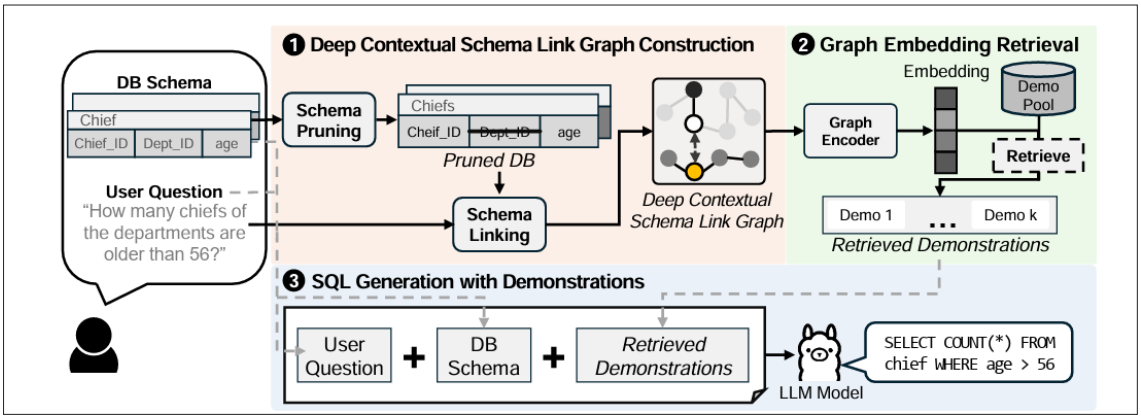
또한 DCG-SQL[17]은 RAG(Retrieval-Augmented Generation) 방식을 활용해 데이터베이스 스키마, 쿼리, 자연어 질의 간의 의미 격차를 줄이고자 하였다. 이러한 접근들은 코드 이해를 단순한 언어 변환 문제가 아닌, 이질적 정보 간 의미 정렬 문제로 재정의한다는 점에서 의미가 크다.

2. 강건성(Robustness)

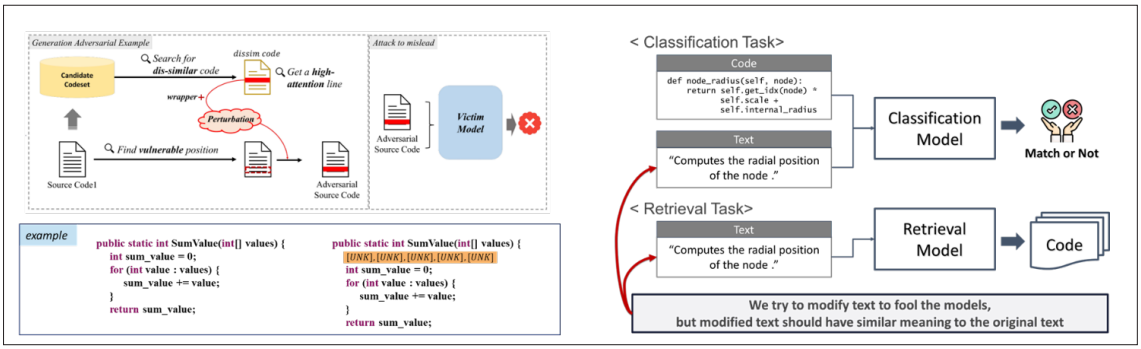
코드 LLM의 강건성은 모델이 문법적 또는 구조적 변형에도 일관된 출력을 유지할 수 있는 능력을 의미한다. 그러나 최근 연구들은 모델이 표면적 교란(perturbation)에



<그림 3> 자연어와 프로그래밍 언어 사이의 의미적 불일치[16]



<그림 4> RAG를 활용한 Text-to-SQL 방법[17]



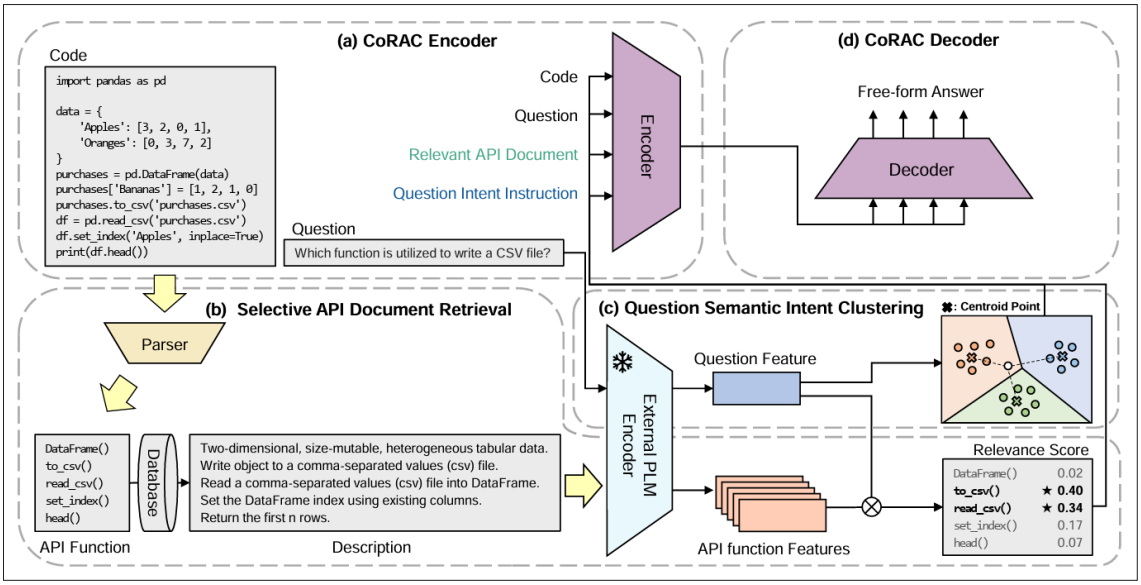
<그림 5> 코드 거대 언어 모델의 취약성[18, 19]

쉽게 영향을 받음을 보여주고 있다. 예를 들어, DIP 연구[18]는 의미가 동일한 코드를 유지한 채 '데드 코드(Dead Code)'를 삽입했을 때 모델의 예측이 크게 흔들리는 현상을 보고하였다. TABS[19]는 변수명 변경, 주석 삽입 등 단순 교란만으로 코드 요약 결과가 크게 달라짐을 보였다.

이러한 결과는 모델이 코드의 의미보다는 문자열 패턴이나 표면 통계적 특징에 과도하게 의존하고 있음을 시사한다. 이에 따라 최근에는 의미 보존형 교란(Semantics-Preserving Perturbation)을 견딜 수 있는 표현 학습과, 코드의 실행 결과를 검증 피드백으로 활용하는 자기 교정(self-verification) 접근이 주목받고 있다.

3. 환각(Hallucination)

코드 LLM이 존재하지 않는 API를 호출하거나 실행 불가능한 코드를 생성하는 문제는 실무 적용에서 매우 치명적이다. 이는 모델이 학습 중 문맥적 제약을 충분히 인식하지 못하거나, 데이터 내 편향된 패턴을 과도하게 일



<그림 6> RAG를 활용한 Code QA 방법[20]

반화할 때 발생한다. 최근 연구들은 이 문제를 해결하기 위해 지식 검색 기반 생성(Knowledge-Augmented Generation)을 도입하고 있다. 예를 들어 RAG 기반 코드 생성은 코드 생성을 수행하기 전, 관련 API 문서나 예시 코드를 검색하여 이를 모델 입력에 포함시키는 방식이다.

또한 API 문서 검색 통합형 코드 QA[20]는 사용자의 질의 의도와 API 호출 의도를 정렬함으로써, 잘못된 코드 제안을 줄이고 신뢰도를 향상시켰다. 이러한 접근은 단순한 텍스트 생성에서 벗어나, 외부 지식과 실행 환경을 결합한 신뢰형 생성(Trustworthy Code Generation)으로 발전하고 있다.

4. 평가(Evaluation)

기존 코드 평가 지표인 BLEU, CodeBLEU 등은 문법적 유사성을 측정할 수는 있으나, 코드의 기능적 동등성(Functional Equivalence)이나 실행 가능성(Executability)을 반영하지 못한다. 이에 따라 최근에는 코드 실행을 포함한 평가 방식이 새롭게 부상하였다. HumanEval[21]은 LLM이 생성한 코드가 주어진 테스트 케이스를 통과할 확률을 측정하는 pass@k 지표를 제안했으며, 이후 대부분의 코드 생성 모델 평가의 표준으로 자리잡았다.

또한 CodeBERTScore[22]는 코드 임베딩 간의 의미적 유사도를 활용해, 코드의 실행 의미를 간접적으로 측정하는 방식을 제시하였다. 향후에는 코드 생성 품질뿐 아니라 보안성, 효율성, 유지보수성 등 다차원 지표를 통합한 평가 체계가 필요할 것으로 보인다.

V. 결론 및 향후 전망

코드 거대 언어 모델(Code LLM)은 이제 단순한 코드 자동화 도구를 넘어, 코드를 이해하고 생성하며 검증할 수 있는 지능적 시스템으로 발전하고 있다.

이러한 기술은 소프트웨어 개발의 효율성과 품질을 동시에 높이며, 인간과 인공지능이 협력하는 새로운 개발 패러다임을 열고 있다. 하지만, 의미 격차(Semantic Gap), 강건성(Robustness), 환각(Hallucination), 평가(Evaluation) 등은 여전히 해결해야 할 핵심 과제로 남아 있다. 향후 연구는 단순한 모델 확장보다, 프로그래밍 언어의 구조적·논리적 의미를 심층적으로 이해하는 학습 체계 구축에 초점이 맞춰질 것이다.

참 고 문 헌

[1] OpenAI, “ChatGPT: Advanced AI Language Model (Version X.Y),” OpenAI Technical Report, 2025.
[2] Anthropic, “Claude: Safe and Steerable Large Language Model,” Anthropic White Paper, 2025.
[3] Google, “Gemini: Next-Generation Multimodal AI System,” Google AI Technical Report, 2025.
[4] Wu et al., “A Survey of Deep Learning Models for Structural Code Understanding,” arXiv preprint arXiv:2205.01293, 2022.
[5] Kanade et al., “CodeBERT: A Pre-Trained Model for Programming and Natural Languages,” Proceedings of EMNLP, 2020.
[6] Husain et al., “CodeSearchNet Challenge: Evaluating the State of Semantic Code Search,” Proceedings of NeurIPS Workshop on ML for Systems, 2019.
[7] Hou et al., “Large Language Models for Software Engineering: A Systematic Literature Review,” arXiv preprint arXiv:2310.12961, 2023.
[8] Guo et al., “GraphCodeBERT: Pre-training Code Representations with Data Flow,” Proceedings of ICLR, 2021.
[9] Ahmad et al., “Unified Pre-training for Program Understanding and Generation,” Proceedings of NAACL-HLT, 2021.
[10] Wang et al., “CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation,” Proceedings of EMNLP, 2021.
[11] Guo et al., “UniXcoder: Unified Cross-Modal Pre-training for Code Representation,” Proceedings of ACL, 2022.
[12] Wang et al., “CodeT5+: Open Code Large Language Models for Code Understanding and Generation,” Proceedings of EMNLP, 2023.
[13] Rozière et al., “Code Llama: Open Foundation Models for Code,” Proceedings of NeurIPS, 2023.
[14] Google DeepMind, “Gemma: Open Models Based on Gemini Research and Technology,” Google AI Technical Report, 2024.
[15] DeepSeek-AI, “DeepSeek-Coder V2: Breaking the Barrier of Code Intelligence,” arXiv preprint arXiv:2406.11931, 2024.
[16] Choi et al., “CodePrompt: Task-Agnostic Prefix Tuning for Program and Language Generation,” Findings of ACL, 2023.
[17] Lee et al., “DCG-SQL: Enhancing In-Context Learning for Text-to-SQL with Deep Contextual Schema Link Graph,” Proceedings of ACL, 2025.
[18] Na et al., “DIP: Dead Code Insertion-based Black-box Attack for Programming Language Model,” Proceedings of ACL, 2023.
[19] Choi et al., “TABS: Efficient Textual Adversarial Attack for Pre-trained NL-Code Model Using Semantic Beam Search,” Proceedings of EMNLP, 2022.
[20] Choi et al., “Integrating Selective API Document Retrieval with Question Semantic Intent for Code Question Answering,” Proceedings of NAACL, 2025.
[21] Chen et al., “Evaluating Large Language Models Trained on Code,” Proceedings of NeurIPS, 2021.
[22] Ren et al., “CodeBERTScore: Evaluating Code Generation with Embedding-based Semantic Similarity,” Proceedings of EMNLP, 2023.

저 자 소 개

최 윤 석

- 2024년 2월 : 성균관대학교 컴퓨터공학과 박사
- 2024년 3월 ~ 2024년 8월 : 한국외국어대학교 교수
- 2024년 9월 ~ 현재 : 성균관대학교 교수
- 주관심분야 : 자연어처리, 대규모 언어 모델, 멀티모달 학습