Journal of KIIT. Vol. 24, No. 7, pp. 253-263, Jul. 31, 2026. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2026.24.7.253

CPG 기반 구조 표현과 검색 근거를 활용한 LLM 검증형 취약점 탐지

신은성*, 정현준**

LLM-based Code Vulnerability Detection using CPG-based Structural Evidence and Retrieval

Eunsung Shin*, Hyunjun Jung**

요 약

함수 단위 텍스트 중심의 LLM 기반 취약점 탐지는 제어 흐름과 데이터 의존성 같은 구조적 단서를 반영하지 못하는 한계가 있다. 본 논문은 코드 속성 그래프(CPG, Code Property Graph)와 그래프 신경망(GNN, Graph Neural Network)에서 학습된 구조 표현을 LLM 검증에 활용하는 취약점 탐지 프레임워크를 제안한다. 내부 평가에서는 LLM-only, GNN-only, GNN+CodeBERT, GNN+retrieval, GraphRAG-V의 F1 점수가 각각 0.655, 0.837, 0.844, 0.850, 0.879로 나타났으며, 공개 보조 평가에서는 Juliet 0.8276, Devign 0.7500의 F1 점수를 기록하였다. 이러한 결과는 CPG 기반 구조 신호가 LLM 기반 취약점 탐지에서 최종 판단 성능 향상에 기여함을 보인다.

Abstract

Text-only LLM-based vulnerability detection often fails to capture structural information such as control flow and data dependency. This paper proposes a vulnerability detection framework that combines Code Property Graph (CPG)-guided signals with Graph Neural Network (GNN) representations and LLM-based reasoning. Internal evaluations report F1 scores of 0.655 for LLM-only, 0.837 for GNN-only, 0.844 for GNN+CodeBERT, 0.850 for GNN+retrieval, and 0.879 for GraphRAG-V, while supplementary public evaluations report 0.8276 on Juliet and 0.7500 on Devign. These results suggest that CPG-guided structural signals can effectively improve LLM-based verification in LLM-based vulnerability detection.

Keywords

software vulnerability detection, code property graph, graph neural network, large language model, retrieval-grounded verification, generalization evaluation

* 국립군산대학교 소프트웨어학과 학사과정
- ORCID: https://orcid.org/0009-0008-2783-9662
** 국립군산대학교 소프트웨어학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0002-6717-1395

· Received: Apr. 07, 2026, Revised: May 21, 2026, Accepted: May 24, 2026

· Corresponding Author: Hyunjun Jung
Dept. of Software at Kunsan National University, 558, Daehak-ro, Kunsan-si, Jeollabuk-do, Republic of Korea
Tel.: +82-63-469-8917, Email: junghj85@kunsan.ac.kr

Ⅰ. 서 론

대규모 언어 모델(LLM, Large Language Model)의 발전은 코드 분석과 보안 자동화의 활용 범위를 크게 확장시켰다[1]. 특히 취약점 탐지 분야에서는 코드 언어모델과 LLM을 이용해 함수 수준의 위험 신호를 판별하거나 취약 여부를 설명하려는 시도가 활발히 이어지고 있다[2]. 이러한 접근은 의미적 단서를 활용할 수 있다는 점에서 주목받고 있으나, 실제 적용성과 신뢰성에 대한 검토는 여전히 중요한 과제로 논의되고 있다[3]. 또한 실제 개발 환경에서는 탐지 결과가 후속 검토와 우선순위 결정에 직접 연결되므로, 모델의 판단이 얼마나 안정적으로 재사용될 수 있는지도 중요하다.

기존 LLM 기반 취약점 탐지 연구의 공통적인 한계는 함수 또는 코드 스니펫 중심의 텍스트 입력에 의존한다는 점이다[3]. 취약점은 제어 흐름, 데이터 의존성, 호출 관계, 자원 해제 순서와 같은 구조적 상호작용 속에서 드러나는 경우가 많다[4]. 그럼에도 불구하고 텍스트 중심 입력은 이러한 구조적 관계를 보존하지 못하며, 더 긴 문맥을 단순히 추가하는 방식은 비관련 정보를 함께 포함하여 판단 안정성을 낮출 수 있다[5].

이러한 한계는 실제 탐지 결과의 해석에서도 문제를 만든다. 예를 들어, 특정 함수 호출의 위험 원인, 취약 조건을 유발하는 데이터 흐름, 입력 검증 누락이 어느 경로를 통해 최종 위험으로 이어지는 경로와 같은 질문에는 단순 텍스트 기반 표현만으로 일관되게 설명하기 어렵다. 따라서 취약점 탐지에서는 표현의 풍부함뿐 아니라, 구조적 상호작용을 명시적으로 유지하는 입력 방식이 필요하다.

또한 취약점 탐지는 단순 분류 문제가 아니라, 왜 해당 코드가 위험한지 설명할 수 있는 근거가 함께 요구되는 영역이다. 따라서 실제 보안 분석 환경에서는 높은 예측 성능뿐 아니라 구조적 근거와 해석 용이성을 함께 제공할 수 있어야 한다. 이러한 요구는 단순한 분류 정확도보다 근거의 일관성과 설명력을 함께 중시해야 함을 의미한다.

특히 실제 코드 검토 과정에서는 탐지 결과가 곧바로 패치 적용 여부나 추가 분석의 우선순위 결정으로 이어지므로, 모델이 어떤 단서를 근거로 판단했는지를 확인할 수 있어야 한다. 구조적 맥락이 결여된 결과는 경고의 수가 많더라도 분석자가 신뢰하기 어렵고, 반대로 구조적 근거가 함께 제시되면 후속 검토에 활용될 수 있다.

그래프 기반 취약점 탐지 연구는 이러한 한계를 보완하기 위한 대안으로 활용되어 왔다[6]. 코드 속성 그래프는 추상 구문 구조, 제어 흐름, 프로그램 의존성을 통합해 표현할 수 있기 때문에 취약점 탐지에 적합한 대표적 중간 표현이다. 그러나 그래프 기반 방법만으로는 API(Application Programming Interface) 의미와 코드 의도를 해석하기 어렵다는 한계가 존재한다[7]. 따라서 구조적 관계를 강하게 포착하는 그래프 기반 표현과, 코드 의미를 해석할 수 있는 언어모델의 결합이 더욱 필요하다. 이러한 결합은 구조적 근거와 최종 판단을 동일한 처리 절차에서 다루기 위한 전제이기도 하다.

이러한 요구는 단순히 그래프 모델과 언어모델을 병렬로 배치하는 수준을 넘어선다. 구조적 근거가 먼저 정리되고, 그 결과가 검색과 의미적 검증 단계로 이어져야 최종 판단의 일관성을 확보할 수 있기 때문이다. 따라서 취약점 탐지 프레임워크는 입력 표현, 구조 학습, 근거 검색, 최종 검증이 순차적 파이프라인 형태로 구성할 필요가 크다.

본 논문은 코드 속성 그래프와 그래프 신경망에서 학습된 구조 표현을 LLM 검증단계에 활용하는 하이브리드 프레임워크를 제안한다. 제안 방법은 원시 코드를 CPG(Code Property Graph)로 변환하고, 구조 임베딩과 기준 예측을 생성한 뒤, 검색 기반 근거와 함께 LLM 검증 단계에 반영하도록 구성하였다. 또한 내부 제거 실험과 공개 보조 평가를 통해 구조 표현, 검색 보강, LLM 검증의 결합 효과와 외부 적용 양상을 함께 점검하였다. 이를 통해 구조 기반 판별과 의미 기반 해석을 동시에 고려하는 취약점 탐지의 실용적 방향을 제시한다고 본다.

즉, 본 논문이 목표로 하는 바는 단일 모델의 절대 성능을 높이는 것에만 있지 않다. 구조 기반 표현으로 취약점 관련 상호작용을 보존하고, 검색 기반 근거와 LLM 검증을 결합함으로써, 최종 판단의 신뢰도와 해석 가능성을 함께 높일 수 있는 실용적

인 탐지 흐름을 제시하는 데 의의가 있다.

II. 관련 연구

2.1 코드 속성 그래프와 그래프 기반 취약점 탐지

코드 속성 그래프는 코드의 구문 구조, 제어 흐름, 프로그램 의존성을 통합하여 구조적 관계를 표현하는 방법으로 널리 활용되어 왔다. 그래프 기반 취약점 탐지 연구는 이러한 표현을 이용하여 함수 내부 및 함수 간 관계를 노드와 엣지 수준에서 학습함으로써, 단순 토큰 기반 모델이 포착하기 어려운 구조적 취약 패턴을 모델링하였다[8].

초기 딥러닝 취약점 탐지가 코드 슬라이스 등을 통해 코드의 의미를 정교하게 반영했다면[8], 이후에는 설명 가능한 연구를 통해 구조 기반 분석의 해석력을 높이는 방향으로 발전했다[7].

이러한 연구는 취약점 판단이 단순한 문자열 패턴이 아니라 관계 기반의 프로그램 행위에 의해 결정된다는 점을 보여주었으나, API 의미와 코드 의도를 해석하는 데에는 한계가 있었다.

2.2 코드 언어모델과 LLM 기반 취약점 탐지

CodeBERT는 자연어와 코드를 함께 사전학습한 대표적인 코드 언어모델로서 코드 분류와 검색 과제에 널리 활용되어 왔다[9]. 이후 GraphCodeBERT는 데이터 흐름 정보를 포함한 구조적 표현을 사전학습에 반영함으로써 코드 이해 능력을 확장하였다[10].

UniXcoder와 CodeT5+와 같은 통합 사전학습 계열 모델도 등장하였으며, 코드 이해와 생성, 분류를 하나의 표현 공간에서 통합하려는 연구가 이어졌다[11][12]. 이와 함께 contextual embedding, code2vec, code2seq, open-vocabulary modeling과 같은 코드 표현 연구가 축적되면서 코드 의미를 더 풍부하게 반영할 수 있는 기반을 넓혀 갔다[13][14].

이후 경로 기반 표현(Path-based representation)과 대규모 코드 코퍼스 기반 초기 자동 취약점 탐지 연구도 이어졌고, 코드 언어모델 기반 접근이 실제 보안 분석 작업에 실제 보안 분석에 적용 가능함을 보여주었다[15][16]. 또한 코드 토큰의 개방형 어휘 문제를 다룬 연구는 프로젝트 코드에서 사전학습 표현이 직면하는 일반화 한계를 이해하는 배경을 제공하였다[17]. 라인 수준 Transformer 기반 접근은 취약점 예측을 더 세밀한 위치 수준으로 확장하였으나, 동시에 구조적 근거와 일반화 성능의 한계도 드러냈다[18].

최근에는 취약점 탐지에 특화된 LLM 평가와 추론 기반 접근(Reasoning)도 활발히 이어지고 있다[2]. 이러한 연구는 코드의 의미를 자연어 수준에서 해석하고, 취약 위험을 설명 가능한 형태로 제시한다는 장점이 있다. 그러나 함수 단위 또는 스니펫 중심의 텍스트 입력만으로는 취약점 판단에 필요한 구조적 단서를 충분히 반영하기 어렵고, 벤치마크에서 얻은 성능이 실제 프로젝트로의 전이 평가 환경에서도 그대로 유지된다고 보기는 어렵다[1].

2.3 구조 표현과 언어모델 결합 연구

최근 연구는 그래프 구조를 LLM 입력과 추론 과정에 결합하는 방향으로 발전하고 있다. CPG-guided 또는 graph-guided LLM 계열 연구는 함수 단위 코드만 제공하는 대신, 취약점 관련 구조를 유지한 슬라이스, 그래프 요약, 검색 근거를 함께 제공함으로써 LLM의 판단 근거를 보강하고자 한다[19].

유사 이웃을 활용한 최근접 이웃 기반(Nearest neighbor) 취약점 탐지 연구 역시, 구조 표현과 검색 기반 근거를 함께 활용하려는 방향과 관련된다[20]. 이러한 접근은 최종 판별 단계에서 비교 가능한 사례를 함께 제공함으로써, 단순 분류보다 근거 중심의 판단을 지원한다.

또한 최근의 맥락 인식 추론 연구(Context-aware reasoning) 연구는 함수 단위 범위를 넘어 인터프로시저 맥락을 활용해야 한다는 점을 강조하고 있으며, 이는 구조 신호와 의미 기반 검증의 결합 필요성을 뒷받침한다[21]. 본 논문은 기존 연구와 달리 CPG/GNN(Graph Neural Network) 구조 표현을 검색 근거 생성과 LLM 검증의 공통 기반으로 활용한다.

III. CPG 기반 LLM 검증형 탐지 방법

3.1 전체 구조

제안 시스템은 데이터 정제 모듈, CPG 생성 및 그래프 변환 모듈, GNN 학습 모듈, 임베딩 및 검색 모듈, LLM 기반 검증 모듈로 포함한다. 시스템의 전체 흐름은 원시 코드로부터 코드 속성 그래프를 구성하고, 이를 그래프 신경망 학습에 적합한 표현으로 변환한 뒤, 구조 임베딩을 검색과 검증에 동시에 활용하는 순서로 진행된다.

그림 1은 제안 시스템의 전체 아키텍처를 나타낸다.

시스템의 핵심 구성 요소는 다음과 같다. 첫째, 데이터 정제 모듈은 원시 코드와 메타데이터의 무결성을 점검하고 학습, 검증, 테스트 분할을 수행한다. 이 단계에서는 코드 중복, 라벨 이상치, 분할 누수와 같은 문제가 후속 학습에 영향을 미치지 않도록 입력 데이터를 정리한다.

둘째, CPG 생성 및 그래프 변환 모듈은 각 샘플을 코드 속성 그래프로 변환하고, 이를 그래프 신경망 학습에 적합한 구조 표현으로 구성한다. 이 과정에서는 노드 유형과 엣지 유형을 명시적으로 유지하여, 코드의 구조적 행위를 관계 형태로 표현한다. 이 단계의 목적은 단순 소스코드 조각을 그대로 전달하는 것이 아니라, 취약점과 직접 관련된 상호작용을 구조화된 형태로 보존하는 데 있다. 따라서 그래프 변환 과정은 이후 구조 학습의 품질을 결정하는 중요한 전처리 단계로 작동한다.

셋째, GNN 학습 모듈은 취약 여부와 관련된 구조 임베딩을 학습하여 기준 예측과 검색 표현을 동시에 생성한다. 즉, GNN은 단순 분류기 역할만 수행하는 것이 아니라, 이후 검색 단계에서 활용될 구조적 표현 공간을 함께 제공한다. 구조 임베딩을 분류와 검색의 공통 표현으로 사용한다는 점은 제안 시스템의 중요한 특징이다. 이를 통해 구조 기반 판별과 검색 기반 근거 수집이 서로 분리된 절차가 아니라, 동일한 구조 표현 위에서 연속적으로 동작하도록 구성할 수 있다.

넷째, 임베딩 및 검색 모듈은 학습된 표현을 바탕으로 유사 샘플을 검색하고, 구조적 근거를 수집한다. 검색된 이웃 샘플은 단순히 유사 코드 예시를 제공하는 것이 아니라, 현재 샘플의 취약 여부를 해석하기 위한 비교 근거로 활용된다. 즉, 유사 샘플의 라벨과 구조 요약은 LLM이 현재 샘플을 독립적으로 판단하는 대신 비교 가능한 사례 기반 판단을 수행하도록 돕는다. 이 과정은 현재 샘플을 구조적으로 유사한 사례 집합 속에서 해석하도록 만든다는 점에서도 의의가 있다.

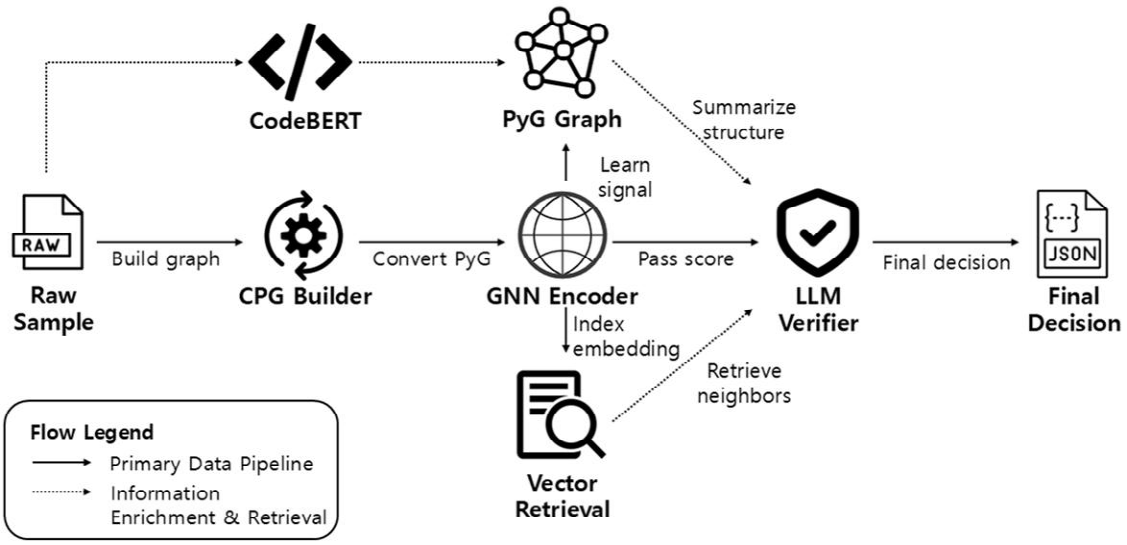

그림 1. 제안 시스템의 전체 아키텍처
Fig. 1. Overall architecture of the proposed system

다섯째, LLM 기반 검증 모듈은 그래프 요약과 검색 근거를 함께 입력받아 최종 취약 여부를 판단한다. 이 단계에서 LLM은 구조 신호와 검색 근거를 종합적으로 해석하여, 기준 예측을 보조적으로 검증하는 역할을 수행한다. 특히 이 모듈은 구조 정보가 없는 상태에서 자유롭게 판단하는 방식보다, 구조 기반 요약과 검색된 이웃 근거를 함께 검토하는 방식으로 구성되었다. 따라서 최종 판별은 단순 언어모델 추론이 아니라 구조 기반 검증에 가까운 흐름으로 뚜렷이 드러난다.

이와 같은 모듈형 구조는 구조 학습, 검색, 의미적 검증을 순차적 파이프라인으로 처리하면서도 각 단계를 독립적으로 최적화할 수 있다는 장점을 가진다. 또한 추후 검색기 교체, 그래프 표현 확장, LLM 교체가 필요할 때에도 전체 시스템을 크게 변경하지 않고 확장할 수 있다. 제안 시스템은 구조적 근거 유지와 의미적 해석 가능성 확보라는 두 요구를 각각 GNN과 LLM의 역할로 분리하면서도 연속된 처리 흐름으로 연결한다. 각 모듈의 개선은 최종 판단 과정 전체에 직접 반영될 수 있다.

3.2 코드 속성 그래프 기반 구조 표현

CPG는 코드의 구문적 구조와 실행 경로, 데이터 의존성을 통합적으로 표현할 수 있기 때문에 취약점 탐지에 적합한 중간 표현이다. 본 논문에서는 각 코드 샘플을 노드 특징, 엣지 연결 정보, 노드 유형, 엣지 유형, 코드 스니펫을 포함하는 그래프 레코드로 구성한다. 이러한 구조는 단순 텍스트 시퀀스와 달리 취약점 관련 상호작용을 관계 기반으로 보존한다.

그래프 표현의 핵심 장점은 취약점과 직접 관련된 구조적 상호작용을 명시적으로 유지할 수 있다는 점에 있다. 예를 들어, 입력 검증 누락, 민감 API 호출 전파, 자원 해제 순서의 불일치와 같은 현상은 토큰 순서만으로는 설명하기 어렵지만, 그래프 구조에서는 노드와 엣지의 관계로 자연스럽게 표현될 수 있다. 이와 같은 구조 표현은 이후 GNN 학습과 검색 기반 근거 생성의 공통 기반이 된다.

또한 그래프 표현은 취약점 근거를 후속 단계에 전달할 수 있는 장점이 있다. 즉, 특정 노드 유형, 특정 엣지 연결, 특정 경로 패턴을 바탕으로 어떤 구조적 상호작용이 판별에 기여했는지를 보다 명시적으로 설명할 수 있다. 이는 후속 검증 단계에서 해석 가능한 근거를 제공하는 데에도 중요하다. 이러한 점에서 구조 표현은 단순한 입력 형식 이상의 의미를 가진다.

3.3 구조 신호 기반 LLM 결합 방식

LLM 결합 단계에서는 단순 코드 텍스트만 제공하지 않고, 앵커 샘플의 그래프 요약과 검색된 이웃 샘플의 구조 정보를 함께 제공한다. 구체적으로는 샘플의 취약 유형, 노드 수와 엣지 수, 주요 노드 및 엣지 유형, 유사도 상위 이웃의 라벨과 구조 요약을 LLM 입력 문맥에 포함한다. 이를 통해 LLM은 취약 코드의 의미를 판단할 때 구조적 단서와 사례 기반 근거를 동시에 참조할 수 있다.

본 논문에서 LLM은 GNN을 대체하는 역할이 아니라, 구조적으로 추출된 구조 표현과 검색 근거를 바탕으로 경계 사례의 최종 판단을 보조한다. 따라서 제안 시스템은 그래프 기반 표현과 언어모델 기반 의미 해석의 상보성을 활용하는 하이브리드 구조이다.

이러한 결합 방식은 LLM을 독립적인 1차 판별기로 사용하지 않는다는 점에서 중요하다. 먼저 구조 표현과 검색 기반 근거를 정리한 뒤, 그 위에서 의미적 판단을 수행하도록 설계함으로써, LLM이 표면적 코드 패턴보다 구조적 단서에 더 강하게 의존하도록 유도할 수 있기 때문이다.

결과적으로 제안 시스템은 구조 표현, 검색, 검증을 각각 분리된 기능으로 배치하면서도 하나의 연속된 의사결정 흐름으로 연결한다. 이는 코드 보안 분석에서 요구되는 구조적 근거와 의미적 해석을 동시에 함께 고려한 설계로 분명히 드러난다.

3.4 제안 방법의 절차

알로리즘 1은 제안 방법의 학습 및 추론 알고리즘을 나타낸다. 이 알고리즘은 코드 속성 그래프 구

성, 그래프 학습, 검색 기반 근거 생성, LLM 검증을 하나의 연속된 절차로 구성한다. 또한 각 단계의 산출물은 다음 단계의 입력으로 활용된다.

알고리즘 1. 제안 방법의 학습 및 추론 알고리즘
Algorithm 1. Procedure of the proposed algorithm

```
Algorithm. Training and inference procedure
Input: raw code set D, label set Y, hyperparameter set H
Output: final prediction and supporting evidence
1: for each sample in D do
    1.1: construct a code property graph
    1.2: convert the graph into a graph-learning
representation
2: split the dataset into train, validation, and test sets
3: train a GNN model
    3.1: obtain structural embedding z
    3.2: obtain prediction score p
4: index train embeddings into the vector database
5: for each test sample do
    5.1: retrieve top-k structural neighbors
    5.2: build verifier context from CPG summary and
retrieved evidence
    5.3: generate the final decision with LLM verification
```

IV. 실험 및 비교 평가

4.1 실험 환경 및 데이터셋

본 연구에서 제안한 CPG 기반 LLM 취약점 탐지 프레임워크의 성능을 평가하기 위해 내부 주요 평가와 공개 보조 평가를 함께 수행하였다. 내부 주요 평가는 PrimeVul 데이터셋을 중심으로 수행하였으며, 비교 대상은 GNN 단독 기준 모델과 CPG/GNN 구조 신호를 반영한 LLM 검증 모델이다[22]. 공개 보조 평가는 Juliet 1.3 C/C++와 Devign 데이터셋을 사용하였다[9][23]. Juliet 평가는 CWE 겹침 기준의 균형 샘플링으로 구성하였고, Devign 평가는 재학습 없이 표본 기반으로 수행하였다.

표 1은 실험 환경을 정리한 것이다. 예시 논문의 형식과 유사하게 운영체제, 프레임워크, 핵심 라이브러리, 주요 모델 정보를 중심으로 간략히 기술하였다.

내부 주요 평가는 구조 기반 기준 모델과 최종 결합 모델의 차이를 확인하는 데 초점을 두었다. 반면 공개 보조 평가는 제안 방법의 외부 적용 가능성을 점검하기 위한 보조 비교축으로 활용하였다. 따라서 본문에서는 내부 주효과와 공개 보조 비교를 구분하여 해석하였다.

표 1. 실험 환경
Table 1. Experimental environment

Operating system	Ubuntu 22.04.1
Framework	Python 3.10.12
Library	PyTorch 2.10.0
	PyG 2.7.0
	Qdrant 1.17.0
CPU	Xeon(R) Silver 4210R
GPU	NVIDIA RTX A6000x2
LLM	GPT-5.2 xhigh

이와 같은 이중 평가 구조는 하나의 수치만으로 전체 방법의 효과를 단정하는 것을 피하기 위한 것이다. 내부 주요 평가는 구성 요소 결합의 효과를 확인하는 데 적합하고, 공개 보조 평가는 외부 데이터셋에 대한 적용 가능성을 방향성 있게 살펴보는 데 적합하다.

4.2 성능 평가 결과 및 분석

내부 주요 평가에서는 GNN 단독 기준 모델과 구조 신호를 반영한 LLM 검증 모델을 비교하였다. 표 2에 제시된 결과를 보면, GNN 단독 기준 모델의 F1 점수 0.837 대비 결합 모델은 F1 점수 0.879를 기록하였다. 절대 성능 차이는 0.042이며, 상대 향상률은 약 5.0% 상승했다.

이 결과는 구조 학습만으로는 구분하기 어려운 경계 사례에서 LLM 기반 의미 해석이 추가적인 보강 효과를 제공함을 보여준다. 특히 제안 방법은 LLM을 독립적인 1차 판별기로 사용하는 것이 아니라, 구조 기반 예측을 보완하는 검증 단계로 활용한다는 점에서 유의하다.

표 2는 F1 중심으로 내부 주요 평가 결과를 제시하며, 결합 모델이 기준 모델보다 더 높은 최종 판별 성능을 보였음을 보여준다. 이는 제안 방법이 특정 보조 구성 하나를 덧붙인 수준이 아니라, 구조 기반 예측 위에 의미 기반 검증을 결합함으로써 전체 판별력을 개선했음을 보여준다. 특히 GNN-only

와 결합 모델의 차이는 구조적 근거 위에 LLM 검증이 추가될 때 경계 사례를 더 안정적으로 처리할 수 있음을 보여준다.

표 2. 내부 주요 평가 결과
Table 2. Primary internal evaluation results

Model	F1-Score
GNN-only model	0.837
GraphRAG-V	0.879

그림 2는 내부 구성요소 제거 실험의 단계별 F1 점수 비교를 나타낸다. LLM-only, GNN-only, GNN+CodeBERT, GNN+retrieval, GraphRAG-V의 다섯 설정을 함께 비교함으로써 구조 표현과 검색, LLM 검증이 순차적으로 결합될 때 최종 성능이 어떻게 향상되는지가 뚜렷이 드러난다.

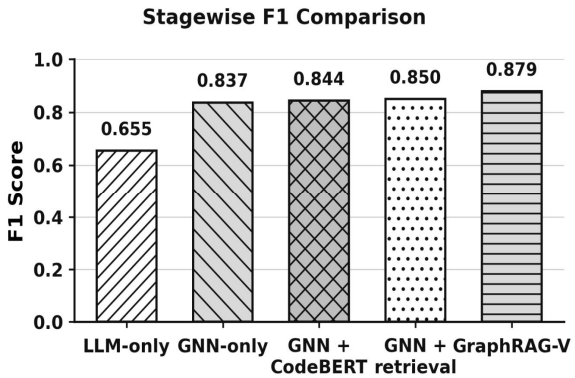

그림 2. 내부 제거 실험의 단계별 F1 점수 비교
Fig. 2. Stagewise F1 comparison in the internal ablation study

표 3의 제거 실험 결과를 보면, LLM-only 설정은 F1 점수 0.655로 가장 낮은 성능을 보였고, 구조 기반 표현을 사용하는 GNN-only 단계에서 가장 큰 폭의 상승이 잘 드러났다. 이후 CodeBERT 결합과 검색 근거 보강(Retrieval grounding)은 각각 추가적인 보강 효과를 제공하였으며, 최종적으로 GraphRAG-V에서 가장 높은 F1 점수 0.879가 드러났다.

이러한 단계별 변화는 구조 표현이 성능 향상의 기본 축을 형성하고 있음을 보여준다. 동시에 검색 기반 근거와 의미적 검증이 추가될수록 최종 성능이 점진적으로 상승한다는 점에서, 제안 방법의 각 구성요소가 누적적으로 기여하고 있음이 뚜렷이 드러난다. 이는 구조 표현, 검색, 의미 기반 검증이 각각 독립적인 부가 기능이 아니라, 최종 판별력을 함께 형성하는 누적적 요소임을 보여준다. 특히 GNN-only에서 GraphRAG-V로 이어지는 최종 향상폭은 구조 표현 위에 검색 보강과 LLM 검증이 누적적으로 더해질 때 가장 안정적인 결과가 나온다는 점을 보여준다. 이는 단계별 결합 전략의 타당성을 보여주는 근거로도 분명히 이어진다. 또한 중간 단계의 점진적 상승은 각 구성요소가 서로 다른 약점을 보완하는 방식으로 작동했음을 보여준다. 예를 들어, CodeBERT 결합은 코드 의미 표현의 보강과 더 관련되고, retrieval 단계는 비교 가능한 사례 기반 근거를 제공한다는 점에서 서로 다른 역할을 가진다. 이 점은 GraphRAG-V의 최종 성능을 해석하는 데도 중요하다.

표 3. 단계별 제거 실험 결과
Table 3. Stagewise ablation results

Stage	Configuration	F1-Score	Delta
1	LLM-Only	0.655	-
2	GNN-only	0.837	+0.182
3	GNN + CodeBERT	0.844	+0.007
4	GNN + retrieval	0.850	+0.006
5	GraphRAG-V	0.879	+0.029

이러한 결과는 구조 표현이 성능 향상의 기본 축을 형성하고, 검색 기반 근거와 LLM 검증이 최종 판별력을 추가적으로 보강한다는 점을 보여준다. 즉, LLM은 단독 판별기로서보다 구조 기반 예측 위에서 의미적 해석을 수행할 때 더 효과적으로 활용될 수 있음을 시사한다.

4.3 기존 연구와의 비교

공개 보조 평가는 제안 방법의 외부 적용 여부를 점검하기 위해 수행하였다. Juliet에서는 F1 점수 0.8276을 기록하였고, Devign에서는 F1 점수 0.7500을 기록하였다. 본 논문에서는 과도한 일반화 주장을 피하기 위해 공개 보조 평가는 F1 점수와 기존 보고된 LLM 계열 최고 F1 점수를 중심으로 비교하였다.

Juliet은 구조적 취약 패턴을 통제된 환경에서 비교할 수 있는 대표적인 공개 테스트 스위트이며, Devign은 그래프 기반 취약점 탐지 연구와 직접적으로 연결되는 대표 벤치마크라는 점에서 보조 비교축으로 유효하다. 따라서 두 데이터셋은 제안 방법의 외부 적용 수준을 방향성 있게 살펴보는 기준으로 활용될 수 있다.

표 4는 Juliet과 Devign에서의 비교 결과를 제시한다. Juliet에서는 기존 LLM 계열 보고치 0.7000 대비 높은 F1 점수를 보였고[3], Devign에서는 현재 확인 가능한 LLM 계열 최고 보고 F1 점수 0.7163보다 높은 수치를 보였다[19]. 다만 평가 프로토콜과 표본 구성은 동일하지 않으므로, 이 비교는 방향성 있는 위치 비교로 해석하는 편이 분명하다.

표 4. 공개 보조 평가의 F1 점수 비교
Table 4. F1 comparison on supplementary public evaluations

Dataset	This study	Reported prior best
Juliet	0.8276	0.7000
Devign	0.7500	0.7163

그림 3은 내부 주요 평가와 공개 보조 평가의 F1 점수 비교를 나타낸다. 내부 주요 평가의 GraphRAG-V 결과와 Juliet, Devign의 F1 점수를 함께 배치함으로써 제안 방법의 전반적인 성능 수준을 비교할 수 있다.

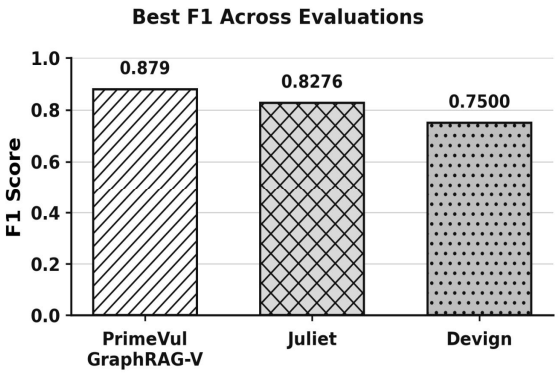

그림 3. 내부 주요 평가와 공개 보조 평가의 F1 점수 비교
Fig. 3. F1 comparison across the internal and supplementary evaluations

그림 4는 공개 보조 평가에서의 F1 점수와 기존 LLM 계열 최고 보고 F1 점수를 비교한 결과를 나타낸다. 두 데이터셋 모두에서 제안 방법이 경쟁력 있는 수준의 성능을 보였음을 확인할 수 있다.

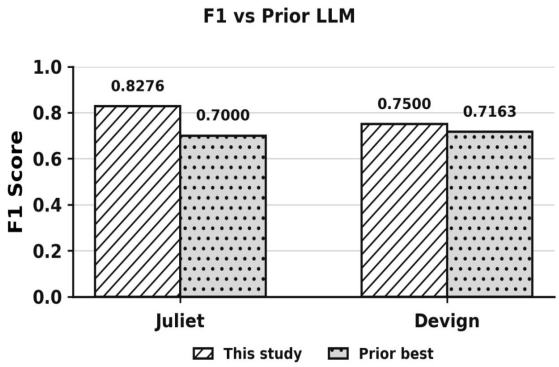

그림 4. 공개 보조 평가와 기존 LLM 계열 결과의 F1 점수 비교
Fig. 4. F1 comparison between this study and prior LLM-based results on supplementary evaluation

이와 같은 비교 결과는 구조 기반 표현과 의미 기반 검증이 공개 데이터셋에서도 일정 수준의 판별력을 유지할 수 있음을 시사한다. 다만 이 비교는 내부 주효과를 직접 대체하는 근거가 아니라, 외부 데이터셋에 대한 보조적 성능 비교로 해석하는 것이 타당하다. 즉, 제안 방법이 공개 데이터셋에서도 경쟁력 있는 수치를 보였다는 사실은 확인되지만, 동일 프로토콜 기반의 직접 비교와는 구분하여 해석해야 한다. 즉, 공개 보조 평가는 절대적인 우열을 단정하기보다 제안 방법이 기존 LLM 계열 결과와 비교해 어느 정도의 경쟁력을 가지는지 확인하는 데 의미가 있다.

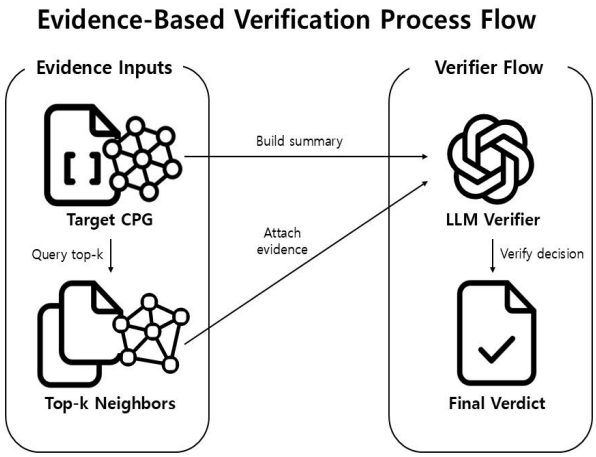

그림 5. CPG 요약, 검색 이웃, LLM 검증 입력의 연결 예시
Fig. 5. Example of the connection among CPG summary, retrieved neighbors, and LLM verifier input

그림 5는 CPG 요약, 검색된 유사 샘플, LLM 검증 입력의 연결 예시를 나타낸다. 제안 방법은 구조 요약과 검색 근거를 함께 제공함으로써, LLM이 단순 텍스트 판단이 아니라 근거 기반 판단을 수행하도록 설계되었다.

Ⅴ. 결론 및 향후 과제

본 논문은 코드 속성 그래프와 그래프 신경망에서 유도된 구조 신호를 LLM과 결합하는 코드 취약점 탐지 프레임워크를 제안하였다. 내부 제거 실험에서는 LLM-only 0.655에서 시작해 GNN-only 0.837, GNN+CodeBERT 0.844, GNN+retrieval 0.850, GraphRAG-V 0.879로 이어지는 단계적 향상 흐름이 확인되었으며, 공개 보조 평가에서는 Juliet 0.8276, Devign 0.7500의 F1 점수를 기록하였다.

이러한 결과는 구조 표현이 성능 향상의 기본 축을 형성하고, 검색 근거와 LLM 검증이 최종 판별력을 추가적으로 보강함을 보여준다. 다만 내부 주요 평가 수치는 비공개 설정에서 분명히 드러난 결과이며, 공개 보조 평가는 제한된 표본과 설정 의존성을 가지므로 외부 타당성 해석에는 더욱 주의가 필요하다. 따라서 결론에서는 내부 주효과와 공개 보조 비교를 구분하여 해석하는 관점이 중요하다.

향후 연구에서는 LLM 호출 수, 입력 토큰 수, 응답 시간 등 추론 비용 지표를 포함하여 실용성을 정량적으로 분석할 예정이다. 또한 실제 서비스 적용을 위해서는 추론 비용과 응답 안정성 분석을 함께 고려한 경량화 전략을 더 구체화해 나가야 한다. 더 나아가 다른 공개 벤치마크와 실제 프로젝트 수준의 평가로 확장하여, 구조 기반 검증 전략의 일반적용 범위를 더 분명히 살펴봐야 한다. 또한 실제 서비스 적용을 위해서는 LLM 호출 수, 입력 토큰 수, 응답 시간 등 추론 비용 지표와 응답 안정성을 함께 분석해야 한다.

References

[1] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep Learning Based Vulnerability Detection: Are We There Yet?", IEEE Transactions on Software Engineering, Vol. 48, No. 9, pp. 3280-3296, Sep. 2022. https://doi.org/10.1109/TSE.2021.3087402.

[2] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, and Y. Chen, "Vulnerability Detection with Code Language Models: How Far Are We?", 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, ON, Canada, pp. 1729-1741, Apr. 2025. https://doi.org/10.1109/ICSE55347.2025.00038.

[3] A. Khare, S. Dutta, Z. Li, A. Solko-Breslin, R. Alur, and M. Naik, "Understanding the Effectiveness of Large Language Models in Detecting Security Vulnerabilities", arXiv preprint arXiv:2311.16169, Oct. 2024. https://doi.org/10.48550/arXiv.2311.16169.

[4] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities", IEEE Transactions on Dependable and Secure Computing, Vol. 19, No. 4, pp. 2244-2258, Jul. 2022. https://doi.org/10.1109/TDSC.2021.3051525.

[5] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks", Advances in Neural Information Processing Systems, Vancouver Canada, Vol. 32, pp. 10197-10207, Dec. 2019. https://proceedings.neurips.cc/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html.

[6] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and Discovering Vulnerabilities with Code Property Graphs", 2014 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, pp. 590-604, May 2014. https://doi.org/10.1109/SP.2014.44.

[7] Y. Li, S. Wang, and T. N. Nguyen, "Vulnerability Detection with Fine-Grained Interpretations", Proc. of the 29th ACM Joint Meeting on European

Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021), Athens, Greece, pp. 292-303, Aug. 2021. https://doi.org/10.1145/3468264.3468597.

[8] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to Represent Programs with Graphs", International Conference on Learning Representations (ICLR 2018), Vancouver, Canada, Apr.-May 2018. https://openreview.net/forum?id=BJOFETxR-.

[9] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection", Proce. 2018 Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, Feb. 2018. https://doi.org/10.14722/NDSS.2018.23158.

[10] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages", Findings of the Association for Computational Linguistics: EMNLP 2020, Online, pp. 1536-1547, Nov. 2020. https://doi.org/10.18653/v1/2020.findings-emnlp.139.

[11] D. Guo, et al., "GraphCodeBERT: Pre-training Code Representations with Data Flow", International Conference on Learning Representations (ICLR 2021), Online, May 2021. https://openreview.net/forum?id=jLoC4ez43PZ.

[12] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "UniXcoder: Unified Cross-Modal Pre-training for Code Representation", Proc. of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Dublin, Ireland, pp. 7212-7225, May 2022. https://doi.org/10.18653/v1/2022.acl-long.499.

[13] Y. Wang, H. Le, A. D. Gotmare, N. D. Q. Bui, J. Li, and S. C. H. Hoi, "CodeT5+: Open Code Large Language Models for Code Understanding and Generation", Proc. of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023), Singapore, pp. 1069-1088, Dec. 2023. https://doi.org/10.18653/v1/2023.emnlp-main.68.

[14] A. Kanade, P. Maniatis, G. Balakrishnan, and K. Shi, "Learning and Evaluating Contextual Embedding of Source Code", Proc. of the 37th International Conference on Machine Learning (ICML 2020), Online, Vol. 119, pp. 5110-5121, Jul. 2020. https://proceedings.mlr.press/v119/kanade20a.html.

[15] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning Distributed Representations of Code", Proc.s of the ACM on Programming Languages, New York, United States, Vol. 3, No. POPL, pp. 1-29, Jan. 2019. https://doi.org/10.1145/3290353.

[16] U. Alon, S. Brody, O. Levy, and E. Yahav, "code2seq: Generating Sequences from Structured Representations of Code", International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, May 2019. https://openreview.net/forum?id=H1gKYo09tX.

[17] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, "Big Code != Big Vocabulary: Open-Vocabulary Models for Source Code", Proc. of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE 2020), Seoul, Korea, pp. 1073-1085, May 2020. https://doi.org/10.1145/3377811.3380342.

[18] J. A. Harer, et al., "Automated Software Vulnerability Detection with Machine Learning", arXiv preprint arXiv:1803.04497, Mar. 2018. https://doi.org/10.48550/arXiv.1803.04497.

[19] M. Fu and C. Tantithamthavorn, "LineVul: A Transformer-Based Line-Level Vulnerability Prediction", Proc. of the 19th International Conference on Mining Software Repositories (MSR 2022), Pittsburgh, PA, USA, pp. 608-620, May 2022. https://doi.org/10.1145/3524842.3528452.

[20] X. Du, M. Wen, J. Zhu, Z. Xie, B. Ji, H. Liu, X. Shi, and H. Jin, "Generalization-Enhanced Code

Vulnerability Detection via Multi-Task Instruction Fine-Tuning", Findings of the Association for Computational Linguistics: ACL 2024, Bangkok, Thailand, pp. 10507-10521, Aug. 2024. https://doi.org/10.18653/v1/2024.findings-acl.625.

[21] Q. Du, X. Kuang, and G. Zhao, "Code Vulnerability Detection via Nearest Neighbor Mechanism", Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, pp. 6173-6178, Dec. 2022. https://doi.org/10.18653/v1/2022.findings-emnlp.459.

[22] Y. Li, T. Zhang, J. Shi, C. Yang, J. He, X. Zhou, J. Jiang, H. Huang, W. B. Leow, Y. Yin, E. L. Ouh, L. K. Shar, and D. Lo, "Beyond Function-Level Analysis: Context-Aware Reasoning for Inter-Procedural Vulnerability Detection", arXiv preprint arXiv:2602.06751, Feb. 2026. https://doi.org/10.48550/arXiv.2602.06751.

[23] P. E. Black, "Juliet 1.3 Test Suite: Changes From 1.2", NIST Technical Note 1995, Gaithersburg, MD, USA, Jun. 2018. https://doi.org/10.6028/NIST.TN.1995.

저자소개

신 은 성 (Eunsung Shin)

2021년 3월 ~ 현재 : 국립군산대학교 소프트웨어학과 학사과정

관심분야 : 보안, LLM, RAG

정 현 준 (Hyunjun Jung)

2008년 3월 : 삼육대학교 컴퓨터과학과(공학사)

2010년 3월 : 숭실대학교 컴퓨터학과(공학석사)

2017년 9월 : 고려대학교 컴퓨터·전파통신공학과(공학박사)

2017년 8월 ~ 2020년 8월 : 광주과학기술원 블록체인인터넷경제연구센터 연구원

2021년 3월 ~ 현재 : 국립군산대학교 소프트웨어학과 교수

관심분야 : 블록체인, 데이터 사이언스, 센서 네트워크, 사물인터넷, 머신러닝