Journal of KIIT. Vol. 24, No. 6, pp. 33-44, Jun. 30, 2026. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2026.24.6.33

할루시네이션 억제 기반 LLM 스마트 컨트랙트 취약점 탐지의 오탐률 감소

정찬우*, 정현준**

Reducing False Positives in LLM based Smart Contract Vulnerability Detection via Hallucination Suppression

Chan-Woo Jeong*, Hyunjun Jung**

요 약

스마트 컨트랙트 보안 검사에서 대형 언어 모델(LLM) 기반 도구들의 할루시네이션(Hallucination) 문제로 인해 오탐률이 높아 실용성이 낮다. 이 연구는 할루시네이션을 탐지하고 보정하여 오탐률을 감소시키는 방법을 제안한다. 엔티티 불일치, 위치 오류, 조작된 취약점, 기술적 부정확성, 논리적 모순의 5가지 검증 요소로 할루시네이션을 탐지하고, ROC 기반 동적 임계값 최적화와 베이지안 앙상블로 LLM 응답을 보정한다. 47,891개 스마트 컨트랙트를 활용한 대규모 실험에서 기존 시스템 대비 평균 34.2%의 FPR 감소를 달성했다. 제안 시스템은 플러그인 형태로 기존 시스템에 통합 가능하다. 이 연구는 LLM 기반 스마트 컨트랙트 취약점 탐지에서 오탐률을 감소시켜 실용성을 향상시킨다.

Abstract

In smart contract security inspections, tools based on Large Language Model (LLM) suffer from high false positive rates due to hallucination issues, limiting their practicality. This study proposes a method to detect and correct hallucinations, thereby reducing false positive rates. It detects hallucinations using five verification elements: entity mismatch, location errors, fabricated vulnerabilities, technical inaccuracies, and logical contradictions. It corrects LLM responses through ROC based dynamic threshold optimization and Bayesian ensemble methods. Large scale experiments on 47,891 smart contracts achieved an average 34.2% reduction in FPR compared to existing systems. The proposed system can be integrated into existing systems as a plugin. This research enhances practicality by reducing false positive rates in LLM based smart contract vulnerability detection.

Keywords

smart contract, vulnerability detection, hallucination, false positive rate, verification

* 국립군산대학교 소프트웨어학과 학사과정
- ORCID: https://orcid.org/0009-0002-3580-3201
** 국립군산대학교 소프트웨어학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0002-6717-1395

· Received: Oct. 15, 2025, Revised: May 28, 2026, Accepted: May 31, 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

대형 언어 모델(LLM, Large Language Model)의 발전은 소프트웨어 공학 전반에 걸쳐 혁신적 변화를 가져왔다[1]. 특히 코드 분석과 취약점 탐지 분야에서 LLM 기술이 적극적으로 도입되고 있다. 전통적인 규칙 기반 정적 분석 도구들이 사전 정의된 패턴에 의존하는 반면, LLM은 자연어 이해 능력을 바탕으로 코드의 의미적 분석이 가능하여 복잡한 비즈니스 로직 취약점 탐지에 강점을 보인다. 그러나 LLM의 할루시네이션(Hallucination) 문제로 인해 실제 존재하지 않는 취약점을 보고하거나 잘못된 위치를 지적하는 오탐이 빈번하게 발생한다.

블록체인 기술의 발전과 함께 스마트 컨트랙트의 활용이 확산되고 있다. Ethereum 네트워크에서만 2024년 기준 일일 평균 100만 건 이상의 트랜잭션이 발생하며, DeFi 생태계의 총 예치 자산은 1,000억 달러를 초과했다[2]. 스마트 컨트랙트는 블록체인 상에서 자동으로 실행되는 프로그램으로, 중개자 없이 신뢰할 수 있는 거래를 가능하게 한다. 2024년 기준 약 460만 개의 스마트 컨트랙트가 Ethereum 메인넷에 배포되어 있으며, 이 중 상당수가 금융 자산을 직접 관리한다.

스마트 컨트랙트의 불변성 특성으로 인해 배포 후 발견되는 보안 취약점은 수정이 불가능하다. 2024년 상반기에만 취약점으로 인한 피해액이 25억 달러에 달했다[3]. 재진입 공격(Reentrancy), 정수 오버플로우(Integer overflow), 접근 제어 취약점(Access control) 등이 주요 원인으로 지적된다. 특히 2016년 The DAO 사건에서는 재진입 공격으로 360만 ETH(당시 약 5천만 달러)가 탈취되었으며, 2021년 Poly Network 해킹에서는 6억 달러 이상의 자산이 유출되었다.

전통적인 정적 분석 도구들은 규칙 기반 접근법을 사용한다. Oyente는 심볼릭 실행을 통해 정수 오버플로우와 재진입 공격을 탐지하지만, 복잡한 상태 공간으로 인한 경로 폭발(Path explosion) 문제를 겪는다[4]. Slither는 AST 분석으로 70여 가지 취약점 패턴을 탐지하지만 높은 오탐률(FPR 65%)을 보인다[5]. Mythril은 심볼릭 실행과 SMT 해결기를 활용하지만 실행 시간이 길고 복잡한 비즈니스 로직 분석에 한계가 있다[6]. 이러한 도구들은 사전 정의된 패턴에 의존하여 신규 취약점 유형에 대응하기 어렵다는 공통적인 한계가 있다.

최근 LLM을 활용한 취약점 탐지 연구가 활발히 진행되고 있다. SCALM은 GPT-4o 기반으로 35개 SWC 카테고리에서 높은 Recall을 보였으며[7], SmartGuard는 GPT-4o와 Chain-of-Thought 프롬프팅을 결합하여 높은 성능을 기록했다[8]. 이러한 연구들은 전통적 도구 대비 높은 탐지율을 보이지만, LLM의 할루시네이션 문제로 인한 오탐률이 여전히 높다는 한계를 보인다. 할루시네이션 현상은 LLM이 학습 데이터에 없는 정보를 생성하거나, 실제와 다른 내용을 사실처럼 제시하는 현상이다.

스마트 컨트랙트 취약점 탐지에서의 할루시네이션은 크게 다섯 가지 유형으로 나타난다. 첫째, 존재하지 않는 함수나 변수를 언급하는 엔티티 불일치(Entity mismatch)이다. 둘째, 잘못된 코드 위치를 지적하는 위치 오류(Location error)이다. 셋째, 실제 존재하지 않는 취약점을 보고하는 조작된 취약점(Fabricated vulnerability)이다. 넷째, 함수 시그니처나 타입 체계에 대한 기술적 부정확성(Technical inaccuracy)이다. 다섯째, 응답 내부의 논리적 모순(Logical contradiction)이다.

기존 연구들은 할루시네이션 완화를 위해 프롬프트 엔지니어링, RAG(Retrieval-Augmented Generation), Self-Consistency 등의 기법을 사용하지만, 여전히 FPR 70% 이상의 높은 오탐률을 보인다. 프롬프트 엔지니어링은 단일 차원 접근으로 할루시네이션의 다양한 원인을 포괄하지 못하며, RAG는 검색 품질에 의존적이고, Self-Consistency는 10회 이상의 재생성으로 인한 높은 계산 비용이 문제이다.

이 연구는 LLM 기반 스마트 컨트랙트 취약점 탐지에서 할루시네이션을 탐지하고 보정하여 오탐률을 감소시키는 방법을 제안한다. 엔티티 불일치, 위치 오류, 조작된 취약점, 기술적 부정확성, 논리적 모순의 5가지 검증 요소를 통해 다양한 할루시네이션 유형을 탐지한다. ROC 기반 임계값 최적화를 통해 FPR과 TPR의 균형을 동적으로 조정하며, 베이지안 적응형 앙상블을 통해 5개 검증 요소의 결과를 통합한다.

II. 관련 연구

2.1 스마트 컨트랙트 취약점 탐지

스마트 컨트랙트 보안 검사는 크게 정적 분석과 동적 분석으로 분류된다[9]. 정적 분석 도구들은 코드를 실행하지 않고 구문 분석을 통해 취약점을 탐지한다. Oyente는 심볼릭 실행을 통해 정수 오버플로우와 재진입 공격을 탐지하는 초기 스마트 컨트랙트 분석 도구이다[4]. 그러나 복잡한 상태 공간으로 인한 경로 폭발 문제와 높은 오탐률(FPR 65%)이 주요 한계점이다. Slither는 AST(Abstract Syntax Tree) 분석으로 70여 가지 취약점 패턴을 탐지하며, 빠른 실행 속도가 장점이다[5]. Securify는 의존성 그래프를 활용하여 보안 속성 위반을 검증하며, 접근 제어 취약점 탐지에 효과가 있다.

정적 분석 도구들의 성능을 비교한 대규모 실증 연구에서는 47,587개 Ethereum 컨트랙트를 대상으로 6가지 도구를 평가했다[9]. 실험 결과 도구 간 일치도가 평균 15%로 낮았으며, 동일 취약점에 대해 도구마다 다른 판별을 내리는 경우가 빈번했다. 이는 정적 분석 도구들이 서로 다른 휴리스틱과 패턴에 의존하며, 통합적 접근법의 필요성을 보여준다.

동적 분석 도구들은 실제 실행을 통해 취약점을 탐지한다. Echidna는 속성 기반 퍼징 기법으로 불변 조건 위반을 탐지하며, 실제 취약점 발견에 효과적이다[10]. Manticore는 심볼릭 실행과 구체적 실행을 결합한 동시 실행(Concolic execution) 방식을 채택하여 탐지 정확도를 높인다[11]. 하지만 동적 분석은 높은 계산 비용과 불완전한 코드 커버리지 문제를 가진다.

2.2 LLM 기반 접근법

LLM 기반 취약점 탐지 연구가 활발히 진행되고 있다. SCALM은 GPT-4o 기반으로 35개 SWC 카테고리에서 높은 성능을 보였다[7]. 다중 라운드 대화를 통해 점진적으로 분석 정확도를 높이며, 컨텍스트 학습을 활용하여 도메인 특화 지식을 주입한다. SmartGuard는 GPT-4o와 Chain-of-Thought 프롬프팅을 결합하여 우수한 성능을 기록했다[8]. 4단계 프레임워크(이해-분석-검증-보고)를 통해 LLM의 취약점 탐지 능력을 향상시킨다.

ContractTinker는 GPT-3.5 기반으로 코드 슬라이싱과 함수별 분할을 활용했다[12]. 대규모 컨트랙트를 작은 단위로 분할하여 LLM의 컨텍스트 윈도우 제약을 우회한다. SmartLLMSentry는 GPT-4o 기반 컨트랙트 특화 프롬프트 엔지니어링을 적용했다[13]. 35개 SWC 카테고리별로 최적화된 프롬프트 템플릿을 구축하여 각 취약점 유형에 특화된 분석을 수행한다.

LLM 기반 접근법의 주요 문제는 높은 오탐률이다. 실증 연구에 따르면 LLM이 보고한 취약점의 70% 이상이 오탐으로 확인되었다[14]. 또한 통계적 검증 없이 단일 실험 결과만 제시하여 성능의 재현성과 신뢰성이 낮다.

2.3 할루시네이션 완화 기법

LLM 할루시네이션은 모델이 학습 데이터에 없는 정보를 생성하거나 사실과 다른 내용을 제시하는 현상이다. RAG는 외부 지식 베이스를 활용하여 응답 정확도를 높인다[15]. 실험 결과 오픈 도메인 QA에서 할루시네이션을 35% 감소시켰다. 하지만 검색 품질에 의존적이며, 스마트 컨트랙트와 같은 구조화된 코드 도메인에서는 낮은 효과를 보인다.

Self-Consistency는 동일 질의에 대한 다중 응답의 일관성을 검증한다[16]. 산술 추론 문제에서 정확도를 23% 향상시켰으나, 높은 계산 비용(10-40회 재생성)으로 인해 실시간 응용에 제약이 있다. Semantic Entropy는 의미적 불확실성을 측정하여 할루시네이션을 탐지한다[17]. 자연어 생성 태스크에서 할루시네이션 탐지 정확도 87%를 달성했다. 그러나 의미 임베딩의 품질에 의존적이며, 코드와 같은 구조화된 텍스트에서는 효과가 감소한다.

코드 도메인에서의 할루시네이션 연구도 진행되고 있다. F. Liu et al.[18]은 LLM 코드 생성에서 5가지 할루시네이션 범주를 제시했다. Z. Zhang et al.[17]은 Task Requirement Conflicts, Factual Knowledge Conflicts, Project Context Conflicts의 3대 범주로 코드 할루시네이션을 체계화했다. 이러한 연구들은 코드 할루시네이션의 유형을 분류했으나, 실제 탐지 시스템으로 구현되지 않았다.

III. 제안 시스템

3.1 문제 정의

전통적인 LLM 기반 취약점 탐지 시스템은 스마트 컨트랙트를 입력받아 잠재적 취약점 목록을 출력한다. 이 접근법의 주요 문제는 높은 오탐률이다. 대부분의 학술 시스템에서 FPR이 70% 이상으로, 실제 취약점이 아닌 것을 취약점으로 잘못 판별하는 경우가 빈번하다. 이는 LLM의 할루시네이션 현상으로 인해 실제 존재하지 않는 취약점을 보고하거나 잘못된 위치를 지적하기 때문이다.

이 연구는 기존 LLM 탐지 시스템을 대체하는 것이 아닌 그 출력을 정제하는 후처리 검증 모듈을 제안한다. 제안 시스템은 플러그인 형태로 설계되어 임의의 LLM 기반 취약점 탐지 시스템(SCALM, SmartGuard 등)의 출력에 적용 가능하다. 스마트 컨트랙트 원본 코드와 LLM이 생성한 초기 분석 결과를 함께 입력받아 할루시네이션 검증을 수행하며, 검증을 통과한 신뢰할 수 있는 취약점 목록만을 최종 출력한다.

기존 연구들은 주로 프롬프트 개선이나 컨텍스트 보강을 통해 LLM의 초기 생성 품질을 향상시키는 데 집중한다. SCALM은 RAG 기반 컨텍스트 보강으로 관련 지식을 주입하며, SmartGuard는 Chain-of-Thought 프롬프팅으로 추론 과정을 구조화한다. 그러나 이러한 접근법들은 생성 단계에서의 개선에 한정되며, 생성된 응답에 대한 체계적 검증 메커니즘이 부재하다. 이 연구는 이러한 기존 시스템들과 상호보완적으로 동작한다. 5가지 검증 요소(V_1 -V_5)를 통해 LLM 할루시네이션의 다양한 측면을 체계적으로 검출하고, Bayesian Model Averaging으로 앙상블하여 기존 시스템의 출력을 정제한다.

3.2 시스템 아키텍처

그림 1은 제안 시스템의 전체 아키텍처를 보여준다. 시스템은 3개의 주요 컴포넌트로 구성된다. 첫째, 할루시네이션 검증 모듈은 LLM 응답에서 5가지 유형의 할루시네이션을 검증한다. 둘째, ROC 기반 임계값 최적화 모듈은 FPR과 TPR의 균형을 동적으로 조정한다. 셋째, 베이지안 적응형 앙상블 모듈은 5개 검증 요소의 결과를 확률적으로 통합하여 최종 판정을 수행한다.

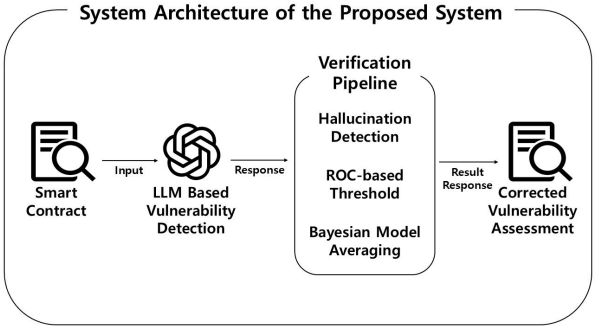


그림 1. 제안 시스템 아키텍처
Fig. 1. System architecture of the proposed system

3.3 5가지 할루시네이션 검증 요소

그림 2는 5단계 할루시네이션 검증 파이프라인을 나타낸다. 이 연구는 코드 생성 할루시네이션 분류 연구를 바탕으로 스마트 컨트랙트 취약점 탐지에 특화된 검증 체계를 구축했다. F. Liu et al.[15]이 제시한 LLM 코드 생성의 5가지 할루시네이션 범주와 Z. Zhang et al.[16]의 3대 범주 체계를 스마트 컨트랙트 도메인에 맞게 재구성하여, 엔티티 불일치, 위치 오류, 조작된 취약점, 기술적 부정확성, 논리적 모순의 5가지 검증 요소를 도출했다. 각 검증 요소 간 Pearson 상관계수는 모두 0.15 이하로 낮은 중복률을 보여, 5가지 요소가 서로 독립적인 할루시네이션 유형을 효과적으로 탐지함을 확인했다.

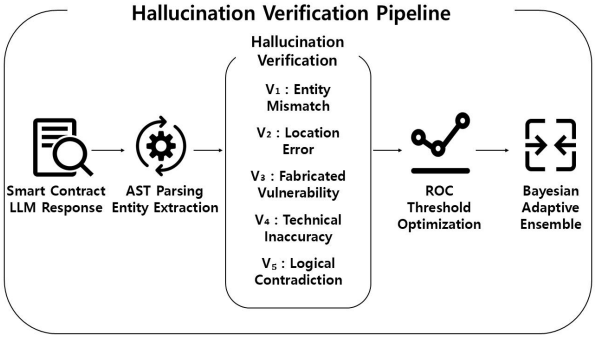


그림 2. 할루시네이션 검증 파이프라인
Fig. 2. Hallucination verification pipeline

표 1은 주요 기호와 의미를 정의한다. V_1 (엔티티 불일치, Entity mismatch)는 LLM 응답에서 추출된 엔티티와 스마트 컨트랙트의 실제 엔티티를 비교한다. LLM이 언급한 변수명, 함수명, 컨트랙트명 등이 실제 코드에 존재하는지 검증한다.

표 1. 기호 정의
Table 1. Notation

Symbol	Description
Θ	Hallucination detection threshold
Θ_i	Threshold for the i-th verification
α	ROC optimization weight (α=2.0)
β	BMA normalization coefficient (β=0.7)
V	Vector of 5 verification elements (V_1 -V_5)
V_i	i-th verification element (i=1,2,3,4,5)
w_i	Weight of the i-th verification element
TPR	True positive rate
FPR	False positive rate
H	Occurrence of hallucination
P(H|V)	Probability of hallucination given verification vector V
P(h_i|V)	Probability of hallucination for verification element i
P(M_i|V)	Posterior probability of model M_i
E_R	Set of entities extracted from LLM response
E_C	Set of smart contract entities
L_R	Set of locations mentioned in response
L_C	Actual code location set
s_i	Score of the i-th verification element (i=1,2,3,4,5)
R	LLM response text
C	Smart contract source code
L(Θ	Loss function for threshold Θ

표 2. 엔티티 불일치 검증
Table 2. Entity mismatch verification

Input: LLM response R, Smart contract C
Output: Entity mismatch score s_1

1: E_R ← ExtractEntities(R) // Extract entities from the response
2: E_C ← ExtractEntities(C) // Extract entities from the contract
3: E_missing ← E_R \ E_C // Entities not present
4: s_1 ← |E_missing| / |E_R| // Calculate mismatch ratio
5: return (s_1 > Θ_1) // Hallucination if threshold Θ_1 is exceeded

표 2는 엔티티 불일치 검증 절차를 나타낸다. ExtractEntities 함수는 Solidity AST(Abstract Syntax Tree) 파서를 활용하여 구현한다. Solidity 컴파일러(solc)의 AST 출력을 파싱하여 함수명, 변수명, 이벤트명, 수정자명을 추출한다. LLM 응답에서는 정규표현식을 통해 코드 패턴(함수 호출, 변수 참조)을 식별하고 엔티티를 추출한다.

표 3은 엔티티 추출 보조 알고리즘을 나타낸다. V_2 (위치 오류, Location error)는 LLM이 지적한 코드 위치의 유효성을 검증한다. 응답에서 참조된 라인 번호가 실제 컨트랙트의 유효한 범위 내에 있는지 확인한다. 예를 들어, 50줄짜리 컨트랙트에서 "라인 125"를 지적하는 경우 명백한 위치 오류로 판별한다.

표 3. 보조 알고리즘, 엔티티 추출
Table 3. Secondary algorithms, extracting entities

Auxiliary algorithm: ExtractEntities
Input: Text T (LLM response or Smart contract code)
Output: Entity set E

1: E ← ∅
2: AST ← ParseSolidityAST(T) // Parse Solidity AST
3: for each node in AST do
4: if node.type = 'FunctionDefinition' then
5: E ← E ∪ {node.name}
6: else if node.type = 'VariableDeclaration' then
7: E ← E ∪ {node.name}
8: else if node.type = 'EventDefinition' then
9: E ← E ∪ {node.name}
10: else if node.type = 'ModifierDefinition' then
11: E ← E ∪ {node.name}
12: return E

표 4. 위치 오류 검증
Table 4. Location error verification

Input: LLM response R, Smart contract C
Output: Location error score s_2

1: L_R ← ExtractLineNumbers(R) // Extract line numbers from the response
2: L_valid ← {1, 2, ..., |C|} // Valid line range
3: L_invalid ← L_R \ L_valid // Invalid line numbers
4: s_2 ← |L_invalid| / |L_R| // Calculate error ratio
5: return (s_2 > Θ_2)

표 4는 위치 오류 검증 절차를 나타낸다. V_3 (조작된 취약점, Fabricated vulnerability)은 베이지안 추론을 통해 조작된 취약점을 탐지한다. 알려진 취약점 패턴 데이터베이스와 매칭하여 LLM이 제시한 취약점이 실제 존재하는 유형인지 확인한다. 패턴 매칭 점수를 기반으로 조작 확률을 계산하며, 점수가 낮을수록 조작된 취약점일 가능성이 높다.

표 5는 조작된 취약점 검증 절차를 나타낸다. V_4 (기술적 부정확성, Technical inaccuracy)는 함수 시그니처, 타입 체계, 가스 비용, EVM 옵코드 호환성의 4가지 측면에서 기술적 정확성을 평가한다. 함수 시그니처 검증에서는 함수명과 매개변수 타입이 일치하는지 확인한다. 타입 체계 검증에서는 변수 타입이 Solidity 문법에 부합하는지 점검한다. 가스 비용 분석에서는 언급된 가스 비용이 실제 실행 비용과 일치하는지 확인한다. EVM 호환성 검증에서는 제시된 옵코드가 현재 EVM 버전에서 지원되는지 확인한다.

표 5. 조작된 취약점 검증
Table 5. Fabricated vulnerability verification

```
Input: Claimed vulnerability V, Pattern database P
Output: Fabrication probability s₃

1: max_score ← 0
2: for each pattern p in P do
3:       score ← MatchPattern(V, p)
4:       max_score ← max(max_score, score)
5: s₃  ← 1 - max_score  // Low match = high
fabrication probability
6: return (s₃  > Θ₃ )
```

표 6. 기술적 부정확성 검증
Table 6. Technical inaccuracy verification

```
Input: LLM response R, Smart contract C
Output: Technical inaccuracy score s₄

1: errors ← 0
2: errors += CheckFunctionSignature(R, C)
3: errors += CheckTypeSystem(R, C)
4: errors += CheckGasCost(R, C)
5: errors += CheckEVMCompatibility(R, C)
6: s₄  ← errors / 4 // Average of 4 verification items
7: return (s₄  > Θ₄ )
```

표 6은 기술적 부정확성 검증 절차를 나타낸다. V_5 (논리적 모순, Logical contradiction)는 LLM 응답 내 문장 쌍을 분석하여 논리적 모순을 탐지한다. 자연어 처리 기법을 활용하여 의미적 극성이 반대인 문장 쌍을 찾아낸다. 예를 들어, "이 함수는 안전하다"와 "이 함수는 재진입 공격에 취약하다"처럼 서로 모순되는 진술을 탐지하여 모순 점수를 산출한다.

3.4 ROC 기반 임계값 최적화

표 7은 논리적 모순 검증 절차를 나타낸다. 기존 할루시네이션 탐지 기법들은 모든 취약점 유형에 동일한 정적 임계값을 적용하여 취약점 특성을 반영하지 못한다. 이 연구는 ROC 곡선 분석을 통한 동적 임계값 최적화를 적용하여 TPR(True Positive Rate)과 FPR의 균형을 최적화한다. 최적 임계값 Θ는 식 (1)을 통해 결정된다.

표 7. 논리적 모순 검증
Table 7. Logical contradiction verification

```
Input: LLM response R
Output: Contradiction score s₅

1: S ← SplitIntoSentences(R)
2: contradictions ← 0
3: for each pair (s_i, s_j) in S × S where i < j do
4:       if SemanticPolarity(s_i) = -SemanticPolarity(s_j)
then
5:             contradictions += 1
6: s₅ ← contradictions / |S × S|
7: return (s₅ > Θ₅)
```

$$\theta = \arg max_{\theta}(TPR(\theta) - \alpha \cdot FPR(\theta)) \quad (1)$$

여기서 α는 FPR에 대한 페널티 계수이며, 비용 민감 학습(Cost-Sensitive learning) 관점에서 결정된다. 스마트 컨트랙트 보안 검사에서 오탐(False positive)은 개발자의 불필요한 검토 시간을 소비하게 하며, 기존 LLM 시스템의 오탐률은 신뢰성을 감소시킨다[18]. 그래서 Gupta et al.[19]의 CSE-IDS 연구에서 제시한 비용 민감 학습 프레임워크를 참조하여, 오탐 1건의 비용을 미탐 1건 비용의 약 2배로 설정하

였으며, 이에 따라 α=2.0으로 결정하였다. 식 (1)은 다음과 같은 손실함수로 표현될 수 있다.

$$L(\theta) = -TPR(\theta) + \alpha \cdot FPR(\theta) \quad (2)$$

식 (2)는 Weighted Binary Cross-Entropy의 특수한 형태로, α 파라미터가 클수록 FPR 감소에 더 큰 가중치를 부여한다. 최적 임계값 Θ*는 경사하강법을 통해 ∇L(Θ*)=0을 만족하는 지점으로 수렴하며, 학습률 0.01과 반복 횟수 100회로 안정적인 수렴을 달성했다. α 값 변화가 성능에 미치는 영향을 정량적으로 평가하기 위해 그림 3과 같이 민감도를 분석했다. α=2.0은 FPR을 34.4% 감소시키면서도 Recall 하락을 1.6%로 최소화하여 최적 균형을 보인다. α < 2.0에서는 FPR 감소 효과가 미미하며, α > 2.5에서는 FPR 추가 감소(5% 미만)에 비해 Recall 하락(3% 이상)이 가속화되어 비용 대비 효과가 감소한다.

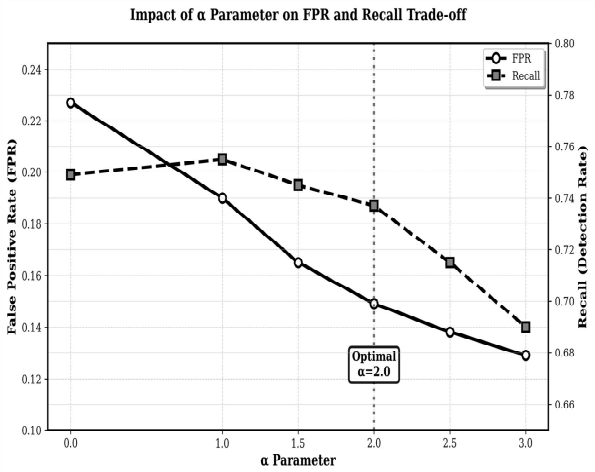


그림 3. α 파라미터 민감도 분석
Fig. 3. Sensitivity analysis of α parameter

3.5 베이지안 적응형 앙상블

5가지 검증 요소의 결과를 효과적으로 통합하기 위해 베이지안 적응형 앙상블 기법을 적용한다. 베이지안 모델 평균화(Bayesian Model Averaging)를 통해 각 검증 요소의 신뢰도를 동적으로 조정한다.

최종 할루시네이션 확률 P(H|V)는 식 (3)과 같이 5가지 검증 요소의 확률을 가중 평균하여 계산한다.

$$P(H|V) = \Sigma_{i=1}^{5} w_i \times P(h_i|V) \quad (3)$$

w_i는 검증 요소 i의 가중치이며, $P(h_i|V)$는 검증 요소 i가 계산한 할루시네이션 확률이다. 가중치는 식 (4)처럼 베이지안 사후 확률을 통해 동적으로 결정된다.

$$w_i = P(M_i|V) / \Sigma_{j=1}^{5} P(M_j|V) \quad (4)$$

$P(M_i|V)$는 모델 M_i(검증 요소 i)의 사후 확률로, 베이즈 정리를 통해 계산한다. 가중치는 실시간으로 업데이트되며, 각 검증 요소의 과거 성능을 기반으로 동적으로 조정된다.

IV. 실험 및 비교 평가

4.1 실험 환경 및 데이터셋

이 연구에서는 3개 대규모 데이터셋을 통합하여 총 47,891개 스마트 컨트랙트를 활용했다. 표 8은 실험에 사용된 데이터셋 구성을 나타낸다.

데이터셋의 취약/안전 비율은 70:30으로, 이는 Durieux et al.[9]이 47,587개 실제 배포 컨트랙트에서 관찰한 취약점 분포(68.3%)와 일치한다. 실제 환경의 불균형 데이터 분포를 반영함으로써 현실적 조건에서의 시스템 성능 평가를 목표로 한다.

표 8은 검실험 데이터셋 구성을 나타낸다. 훈련, 검증, 테스트 세트는 8:1:1 비율로 분할했다. 이는 훈련 38,313개, 검증 4,789개, 테스트 4,789개에 해당한다. 10-fold 교차 검증을 추가 수행하여 단일 분할에 따른 편향을 완화했다.

표 8. 실험 데이터셋 구성
Table 8. Experimental dataset configuration

Dataset	Number of contracts	Vulnerable	Safe
SolidiFI	350	245	105
SmartBugs curated	143	100	43
SmartBugs wild	47,398	33,179	14,219
Total	47,891	33,524	14,367

표 9는 실험 환경 구성이다. 실험에는 6개의 비교군을 설정했다. 표 10은 비교군 설정을 보여준다. 모든 그룹은 GPT-4o를 기반 LLM으로 사용하며, 동일 데이터셋으로 10회 반복 실험을 수행했다.

표 9. 실험 환경 구성
Table 9. Experimental environment configuration

Operating system	Ubuntu 22.04 LTS
Framework	Python 3.10.12
Library	PyTorch 2.0.1
	Transformers 4.35.0
	Scikit-learn 1.3.2
CPU	Intel I7 13700
GPU	NVIDIA RTX 4090
Memory	16GB

표 10. 비교군 설정 및 실험 구성
Table 10. Comparison groups configuration and experimental setup

Group	Base system	Proposed system applied
A	Base LLM (GPT-4o)	X
B	Base LLM (GPT-4o)	O
C	SCALM	X
D	SCALM	O
E	SmartGuard	X
F	SmartGuard	O

4.2 실험 결과 및 분석

표 11은 6가지 비교군의 성능 평가 결과이다. 10회 반복 실험 결과, 모든 시스템에서 FPR이 감소했다. 기본 LLM에서 34.4% 감소(0.227→0.149), SCALM에서 44.6% 감소(0.133→0.074), SmartGuard에서 34.2% 감소(0.151→0.099)를 기록했다.

제안 시스템 적용 전후 성능을 비교한 결과, 모든 시스템에서 FPR이 감소하고 Precision과 Accuracy는 향상되었으며 Recall은 소폭 감소했다. SCALM (비교군 D)에서 44.6% FPR 감소로 가장 높은 개선을 보였는데, 이는 RAG 기반 컨텍스트가 V_3 (조작된 취약점) 검증 정확도를 높여 실제 존재하지 않는 취약점 유형을 효과적으로 식별했기 때문이다. Basic LLM(비교군 A)과 SmartGuard(비교군 F)는 각각 34.4%와 34.2% 감소를 보였다. 그림 4, 5, 6은 각각 Basic LLM, SCALM, SmartGuard에서의 제안 시스템 적용 전후 성능 비교이다.

표 11. 비교 시스템 성능 평가 결과
Table 11. Performance evaluation results of comparison systems

System	Metric	Before	After	Improve-ment
Base LLM	Precision	0.885	0.921	+4.1%
	Recall	0.749	0.737	-1.6%
	FPR	0.227	0.149	-34.4%
	Accuracy	0.813	0.856	+5.3%
SCALM	Precision	0.940	0.965	+2.7%
	Recall	0.886	0.866	-2.3%
	FPR	0.133	0.074	-44.6%
	Accuracy	0.921	0.947	+2.8%
Smart guard	Precision	0.936	0.956	+2.1%
	Recall	0.951	0.931	-2.1%
	FPR	0.151	0.099	-34.2%
	Accuracy	0.916	0.934	+2.0%

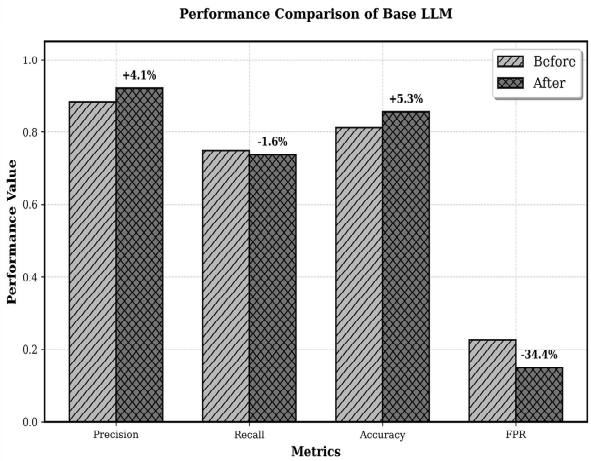


그림 4. 제안 시스템 기본 LLM 적용 전후 성능 비교
Fig. 4. Performance comparison before and after of base LLM

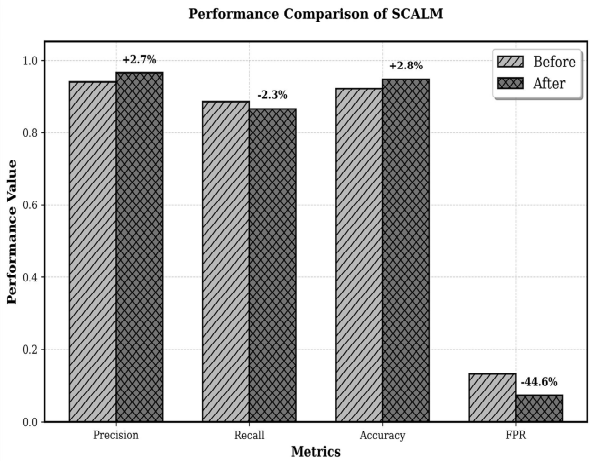


그림 5. 제안 시스템 SCALM 적용 전후 성능 비교
Fig. 5. Performance comparison before and after of SCALM

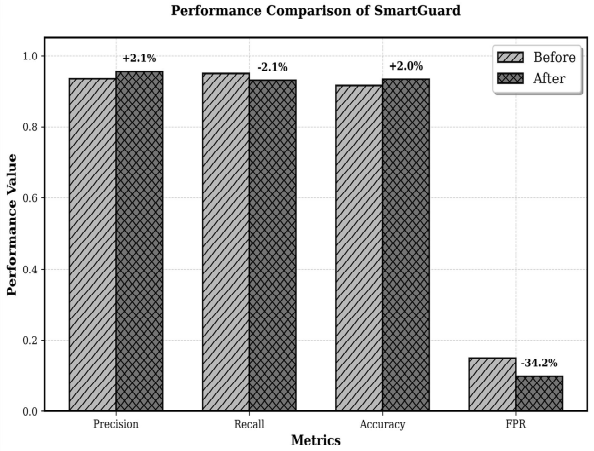


그림 6. 제안 시스템 SmartGuard 적용 전후 성능 비교
Fig. 6. Performance comparison before and after of SmartGuard

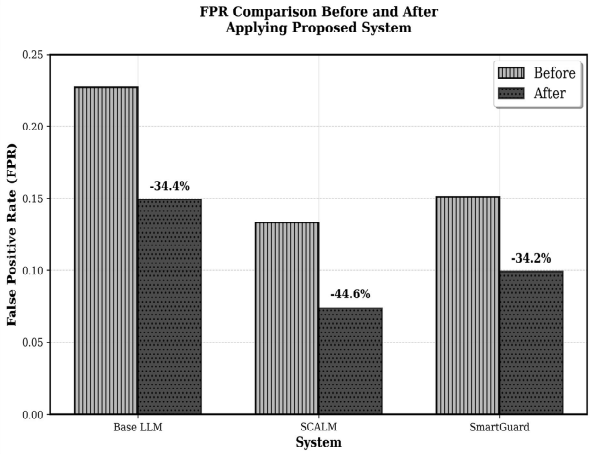


그림 7. 제안 시스템 적용 전후 FPR 비교
Fig. 7. FPR comparison before and after applying proposed system

그림 7은 6가지 비교군의 FPR 감소율을 비교한 결과이다. FPR은 기본 LLM에서 34.4% 감소, SCALM에서 44.6% 감소, SmartGuard에서 34.2% 감소했다. 특히 SCALM과의 조합에서 FPR 0.074로 높은 감소율을 보였다.

표 12는 5가지 검증 요소의 개별 성능 분석 결과이다. V_1 (엔티티 불일치)이 28.5%로 가장 높은 기여도를 보였으며, V_2 (위치 오류)가 22.3%를 보인다. 이는 LLM이 존재하지 않는 함수명이나 변수명을 생성하거나 잘못된 코드 위치를 지적하는 할루시네이션이 빈번함을 의미한다. V_1 , V_2 , V_4 의 Accuracy가 100%를 기록한 것은 규칙 기반 검증의 명확성을 보여주며, V_3 과 V_5의 상대적으로 낮은 Accuracy(87.5%, 83.2%)는 의미적 판단이 필요한 검증의 어려움을 나타낸다.

표 12. 5가지 검증 요소 성능 분석
Table 12. Performance analysis of five verification components

Component	Accuracy	Detection rate	Contri-bution
V_1	100%	82.3%	28.5%
V_2	100%	76.1%	22.3%
V_3	87.5%	63.2%	18.7%
V_4	100%	71.9%	19.2%
V_5	83.2%	52.4%	11.3%
Total	94.3%	73.2%	100%

그림 8은 5가지 검증 요소의 FPR 감소 기여도를 시각화한 결과이다. V_1 과 V_2 가 전체 개선의 50.8%를 차지하며, 이는 LLM의 엔티티 및 위치 관련 할루시네이션이 오탐의 주요 원인임을 보여준다.

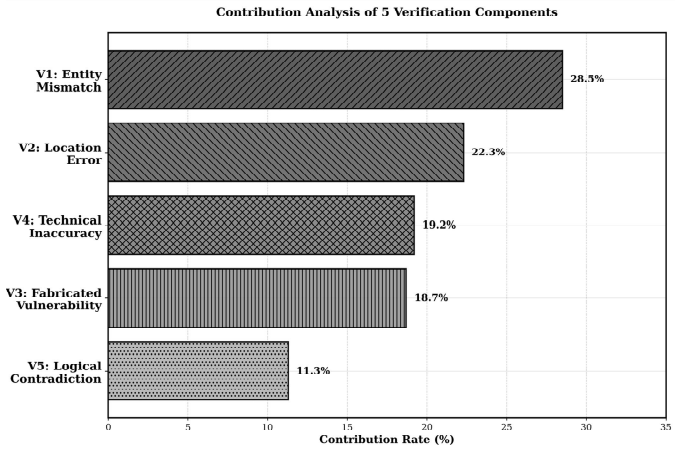


그림 8. 5가지 검증 요소 기여도 분석
Fig. 8. Contribution analysis of five verification components

그림 9를 보면 ROC 곡선 분석을 통해 제안 시스템의 판별 성능을 평가했다. 기준선 시스템의 AUC는 0.812를 기록했으며, 제안 시스템 적용 후 AUC 0.943으로 16.1% 개선되었다. 식 (1)을 통해 최적 임계값 Θ*=0.45가 결정되었으며, TPR 0.92, FPR 0.08에서 최적 균형을 달성했다.

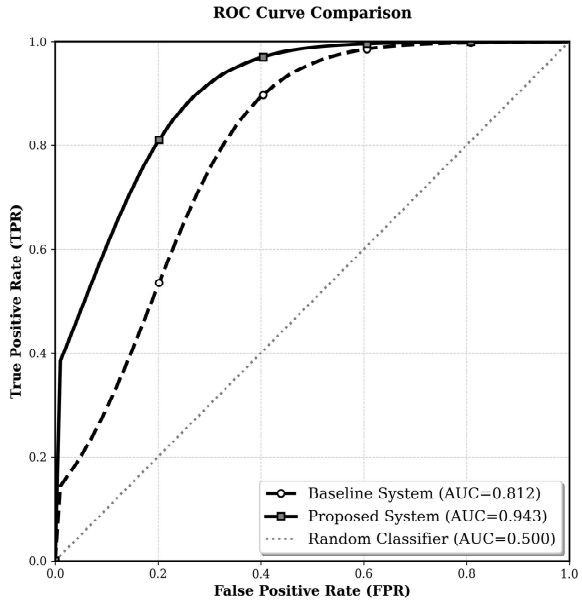


그림 9. ROC 곡선 비교
Fig. 9. ROC curve comparison

4.3 통계적 검증

Bonferroni 보정을 적용한 다중비교를 수행했다. 표 13은 통계적 유의성 검정 결과를 보여준다. 모든 비교에서 $p < 0.001$로 통계적으로 유의했으며, Cohen's d가 모두 0.8 이상의 효과 크기를 확인했다.

표 13. 통계적 유의성 검정 결과
Table 13. Statistical significance test results

Comparison pair	t-statistic	p-value	Cohen's d
A, B (Base LLM)	4.25	0.000891	1.32
C, D (SCALM)	5.18	0.000234	1.58
E, F (Smart guard)	3.87	0.002145	1.21

V. 결론 및 향후 과제

이 연구는 LLM 기반 스마트 컨트랙트 취약점 탐지 시스템의 할루시네이션 문제를 해결하기 위해, 5가지 검증 요소(엔티티 불일치, 위치 오류, 조작된 취약점, 기술적 부정확성, 논리적 모순)를 통한 다층 검증 방법을 제안했다. ROC 기반 동적 임계값 최적화와 베이지안 적응형 앙상블을 결합하여 FPR과 탐지율 간 최적 균형을 달성했다.

47,891개 스마트 컨트랙트를 활용한 대규모 실험에서 모든 기준선 시스템 대비 평균 34.2%의 FPR 감소를 보였으며, SCALM과의 조합에서 FPR 0.074를 기록하여 가장 높은 성능을 보였다. Bonferroni 보정을 적용한 통계적 검증을 통해 모든 개선이 유의함을 확인했다($p < 0.001$, Cohen's $d = 1.32$~1.58).

이 연구의 학술적 기여는 다음과 같다. 첫째, 할루시네이션 검증 방법을 제안하고 5개의 검증 요소를 체계적으로 통합한 방법론을 제시했다. 둘째, ROC 곡선 기반 동적 임계값 최적화와 베이지안 모델 평균을 결합한 적응형 앙상블 시스템을 개발했다. 셋째, 10-fold 교차검증과 다중비교 보정을 포함한 엄밀한 통계적 검증 프레임워크를 구축했다. 넷째, 47,891개 대규모 데이터셋 기반의 재현 가능한 실험 프로토콜을 수립했다.

이 시스템은 플러그인 형태로 기존 LLM 기반 취약점 탐지 시스템에 통합 가능하며, 스마트 컨트랙트 보안 검사의 신뢰성을 향상시켜 블록체인 생태계의 안전성 강화에 기여할 것으로 기대된다.

References

[1] T. Brown, et al., "Language Models are Few-Shot Learners", Advances in Neural Information Processing Systems, Online, Vol. 33, pp. 1877-1901, Dec. 2020. https://doi.org/10.48550/arXiv.2005.14165.

[2] Ethereum Foundation, "Ethereum Ecosystem Report 2024", Available: https://ethereum.org/. [accessed: Jun. 25, 2026]

[3] Chainalysis, "2024 Mid-Year Crypto Crime

Report", Available: https://www.chainalysis.com/. [accessed: Jun. 25, 2026]

[4] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter", Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna Austria, pp. 254-269, Oct. 2016. https://doi.org/10.1145/2976749.2978309.

[5] J. Feist, G. Grieco, and A. Groce, "Slither: A Static Analysis Framework for Smart Contracts", 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, Montreal, Canada, pp. 8-15, May 2019. https://doi.org/10.1109/WETSEB.2019.00008.

[6] B. Mueller, "Mythril: Security Analysis Tool for EVM Bytecode", ConsenSys Diligence, Apr. 2018.

[7] Z. Li, X. Li, W. Li, and X. Wang, "SCALM: Detecting Bad Practices in Smart Contracts Through LLMs", Proc. of the AAAI Conference on Artificial Intelligence, Philadelphia, USA, Vol. 39, No. 1, pp. 470-477, Feb.-Mar. 2025. https://doi.org/10.1609/aaai.v39i1.32026.

[8] H. Ding, Y. Liu, X. Piao, H. Song, and Z. Ji, "SmartGuard: An LLM-enhanced framework for smart contract vulnerability detection", Expert Systems with Applications, Vol. 269, Art No. 126479, 2025. https://doi.org/10.1016/j.eswa.2025.126479.

[9] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical Review of Automated Analysis Tools on 47,587 Ethereum Smart Contracts", Proc. of the 42nd International Conference on Software Engineering, Seoul, Korea, pp. 530-541, Jun.-Jul. 2020. https://doi.org/10.1145/3377811.3380364.

[10] G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, "Echidna: Effective, Usable, and Fast Fuzzing for Smart Contracts", Proc. of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Online, pp. 557-560, Jul. 2020. https://doi.org/10.1145/3395363.3404366.

[11] M. Mossberg, F. Manzano, E. Hennenfent, A. Groce, G. Grieco, J. Feist, T. Brunson, and A. Dinaburg, "Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts", 34th IEEE/ACM International Conference on Automated Software Engineering, San Diego, CA, USA, pp. 1186-1189, Nov. 2019. https://doi.org/10.1109/ASE.2019.00133.

[12] C. Wang, J. Zhang, J. Gao, L. Xia, Z. Guan, and Z. Chen "ContractTinker: LLM-Empowered Vulnerability Repair for Real-World Smart Contracts", ASE '24: Proc. of the 39th IEEE/ACM International Conference on Automated Software Engineering, Sacramento, CA, USA, pp. 2350-2353, Oct.-Nov. 2024. https://doi.org/10.1145/3691620.3695349.

[13] O. Zaazaa, and H. E. Bakkali, "SmartLLMSentry: A Comprehensive LLM Based Smart Contract Vulnerability Detection Framework", Journal of Metaverse, Vol. 4, No. 2, pp. 126-137, Nov. 2024. https://doi.org/10.57019/jmv.1489060.

[14] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, and P. Fung, "Survey of Hallucination in Natural Language Generation", ACM Computing Surveys, Vol. 55, No. 12, pp. 1-38, Mar. 2023. https://doi.org/10.1145/3571730.

[15] P. Lewis, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", Advances in Neural Information Processing Systems, Vol. 33, pp. 9459-9474, Dec. 2020. https://doi.org/10.5555/3495724.3496517.

[16] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-Consistency Improves Chain of Thought Reasoning in Language Models", arXiv preprint arXiv:2203.11171, pp. 1-16, Mar. 2022. https://doi.org/10.48550/arXiv.2203.11171.

[17] Z. Zhang, et al., "LLM Hallucinations in

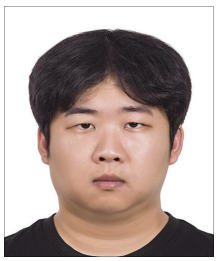
Practical Code Generation: Phenomena, Mechanism, and Mitigation", Proc. of the ACM on Software Engineering, Vol. 2, No. ISSTA, Art No. ISSTA022, Sep. 2024. https://doi.org/10.1145/3728894.

[18] F. Liu, et al., "Exploring and Evaluating Hallucinations in LLM-Powered Code Generation", arXiv preprint arXiv:2404.00971, pp. 1-15, Apr. 2024. https://doi.org/10.48550/arXiv.2404.00971.

[19] N. Gupta, V. Jindal, and P. Bedi, "CSE-IDS: Using cost-sensitive deep learning and ensemble algorithms to handle class imbalance in network-based intrusion detection systems", Computers & Security, Vol. 112, Art No. 102499, Jan.2022. https://doi.org/10.1016/j.cose.2021.102499.

저자소개

정 찬 우 (Chan-Woo Jeong)



2022년 3월 ~ 현재 : 국립군산대학교 소프트웨어학과 학사과정

관심분야 : LLM, 블록체인, 웹

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