Journal of KIIT. Vol. 24, No. 4, pp. 11-17, Apr. 30, 2026. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2026.24.4.11

대규모 언어모델의 반복 자율 검증 구조를 활용한 소스 코드 취약점 패치 검증 에이전트 설계

조수현*, 김준범**, 이승현***

Design of a Source Code Vulnerability Patch Verification Agent using an Iterative Self-Verifying Architecture with Large Language Models

Soohyun Jo*, Junbeom Kim**, and Seunghyun Lee***

요 약

최근 프로그램 소스 코드의 취약점 탐지 등 정적 분석 도구로 대규모 언어모델(LLMs, Large Language Models)을 활용하는 연구가 활발하다. LLMs는 전통적인 정적 분석 도구와 달리, 코드의 맥락을 활용한다. 그러나 일반적인 LLMs는 질의 - 응답으로 이어지는 단순한 수준으로 자기 검증과 자기 수정 절차가 부족하다. 이에 본 연구에서는 반복 자율 검증 루프 구조에 LLMs를 활용할 때 효용성을 확인한다. 연구 설계로 LLMs 기반의 검증자 중심 반복 자율 검증 구조의 자율형 검증 에이전트를 구성하여 소스 코드의 취약점에 대한 탐지·수정·검증 기능을 수행한다. 검증 과정에서 생성된 피드백을 다시 입력으로 활용하는 구조를 구성하고, 반복에 따른 성과 변화를 측정하여 개선 한계선을 측정한다. 이를 통해 본 연구는 소스 코드 분석에서 검증자 중심 반복 자율 검증 구조의 효과를 확인하고, 최적 반복 기준을 제시한다.

Abstract

Recently, there has been active research on the application of Large Language Models (LLMs) as static analysis tools for source code vulnerability detection. However, LLMs in general lack self-verification and self-correction mechanisms, which are only found in their question-and-answer interaction processes. This paper, therefore, investigates the effects of employing LLMs in an iterative self-verifying loop. We design a self-verifying agent and construct a verifier-centric iterative self-verification loop structure that utilizes LLMs to perform vulnerability detecting, fixing, and verifying source code. We also implement the self-verifying agent and conduct experiments to verify the efficiency of using the verifier-centric iterative self-verification loop in source code analysis and to determine the optimal iteration based on our proposed criterion.

Keywords

large language models, LLM, LLMs, vulnerability detection, self-revision, self-verification

* 고려대학교 컴퓨터학과 박사과정(교신저자)
- ORCID: https://orcid.org/0000-0002-1828-2814
** 이글루코퍼레이션 선임연구원
- ORCID: https://orcid.org/0000-0003-1779-9173
*** ㈜노바프로토콜 대표, 건국대학교 겸임교수
- ORCID: https://orcid.org/0000-0002-3945-5075

· Received: Jan. 08, 2026, Revised: Feb. 25, 2026, Accepted: Feb. 28, 2026

· Corresponding Author: Soohyun Jo
Dept. of Computer Science and Engineering, Korea University, 145 Anam-ro, Seongbuk-gu, Seoul, Seoul, 02841, Korea
Tel.: +82-2-3290-3206, Email: soohyunjo@korea.ac.kr

Ⅰ. 서 론

최근 대규모 언어모델(LLMs, Large Language Models)이 소스 코드의 취약점 탐지 등 정적 분석의 영역에 활용되고 있다. LLMs는 전통적인 정적 분석 도구들과 다르게 코드를 문맥 차원에서 분석을 수행한다[1]-[4]. 하지만 기본적으로 LLMs는 자기 검증이나 결과 수정 기능이 없이 단일 질의 - 응답 수준에 머물러 있다[1][5].

학계에서는 LLMs의 코드 분석 및 검증 활용에 대한 논의가 활발하다.

N. Shinn et al.[6]은 학습 이력을 스스로 반성하고 기억하는 구조의 언어 에이전트를 제안하여 반복 수행 횟수가 증가할수록 성능이 향상됨을 확인하였다. 이 연구는 LLMs가 피드백을 기반으로 스스로 발전하는 자율적 에이전트로 진화할 수 있음을 보여준다. A. Kumar et al.[7]은 실제 개발 환경에서 수행된 에이전트와 개발자의 협업 연구를 통해, 에이전트가 단일 시도(One-shot)로 문제를 해결하기보다 작업을 세분화하여 점진적으로 해결할 때 성공률이 높다는 점을 확인하였다. 특히, 에이전트의 결과물의 단순 수용보다 능동적인 피드백을 통해 수정과 검증을 반복하는 과정이 복잡한 소프트웨어 결함을 해결하는 데 필수적임을 강조하였다.

일부 연구는 프로그램 실행 과정을 단계적으로 추적하며 인간과 유사한 방식으로 검증을 수행하였다[1]. 반면, 다른 연구에서는 테스트 단계에서 발생한 오류를 모델이 스스로 식별하고 수정하도록 설계하였다[2]. X. Wang et al.[8]의 Self-Consistency 연구에서는 동일한 문제에 대해 여러 추론 경로를 샘플링하고 상호 증거를 찾아 답변을 더욱 신뢰할 수 있는 기술을 제안했다. 이러한 선행연구에서 반복적으로 출력을 점검하고 수정하는 루프 구조를 통해 LLMs의 결과 품질 향상에 도움이 된다는 것을 확인하였다. 이는 수행자(Executor)와 검증자(Validator) 구조의 이론적 근거를 제공한다.

S. Choi et al.[9]은 LLMs를 이용하여 코드의 품질을 향상할 수 있음을 확인하였으며, S. Lee et al.[10]은 GPT 모델의 분석 결과로 신뢰도와 타당도 측면을 평가, 검증 체계의 중요성을 강조하였다[4].

본 연구에서는 기존의 선행연구를 바탕으로 검증자 중심 반복 자율 검증 루프 구조를 제안하여, 반복 횟수에 따라 취약점의 탐지, 수정의 성과가 어떻게 변화하는지 실험을 통해 확인하였다.

Ⅱ. 선행 연구

N. Shinn et al.[6]은 스스로 반성하고 학습 이력을 기억하는 언어 에이전트를 제안했다. 이들은 모델이 과제 수행 중 마주한 오류나 한계는 일종의 반성문으로 작성하여 저장하며, 이를 통해 보다 나은 결정을 내리는 Reflexion이라는 구조를 제안했다. 이를 언어 기반 강화학습(Verbal reinforcement learning)이라고 정의했다.

그림 1은 Reflexion의 동작을 보여준다. Actor는 행동을 결정하고, Evaluator는 결정을 평가한다. 이어서 Self-reflection는 평가 결과를 바탕으로 언어적인 피드백을 생성한다. 피드백은 행위 기억 장소(Experience memory)에 저장되어 다음 시행에서 Actor가 참고하게 된다.

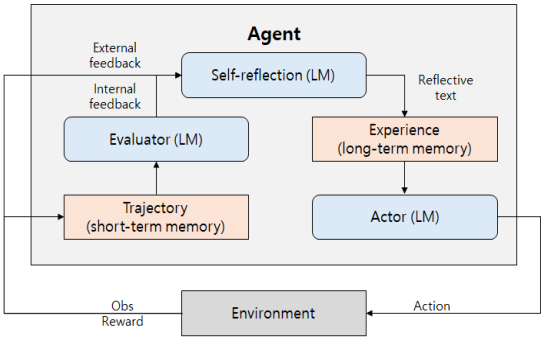

그림 1. Reflexion의 다이어그램
Fig. 1. Diagram of Reflexion

Reflexion은 ALFWorld, HotPotQA, HumanEval과 MBPP 데이터셋을 활용한 코드 생성 실험에서 학습 효과를 검증하였다.

다만 한계점으로 제한된 기억의 크기에 따라 장기적 맥락 유지가 어렵고, 일부 환경에서는 탐색 다양성의 부족으로 개선 효과가 제한적이었다.

그림 2는 이전 연구에서 제안된 탐지 - 수정 - 검증(DFV, Detect - Fix - Verify) 에이전트의 동작이다. DFV는 반복 초기 구간에서 취약점 탐지와 개선에 효과를 보여주었다[11].

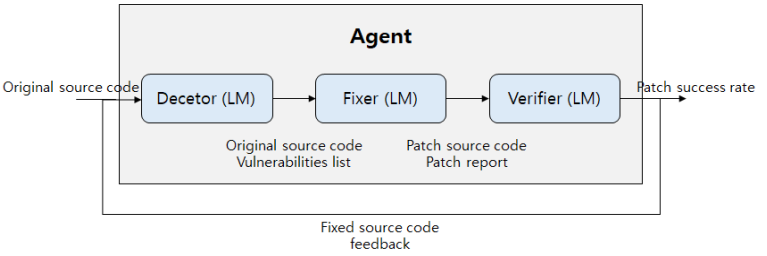

그림 2. 탐지 - 수정 - 검증(DFV) 루프 에이전트
Fig. 2. Detect - Fix - Verify (DFV) agent

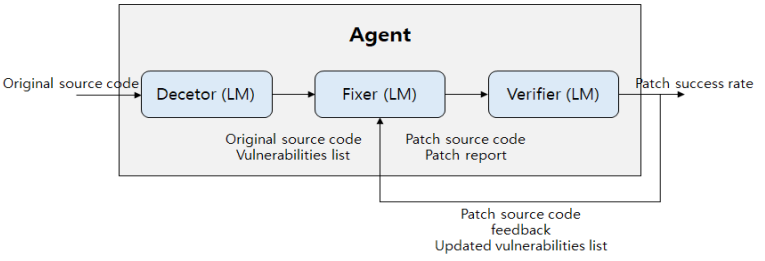

그림 3. 제안된 자율형 검증 에이전트의 아키텍처
Fig. 3. Architecture of the proposed autonomous verification agent

본 연구는 DFV에서 제안된 탐지 - 수정 - 검증 루프 에이전트를 기본으로 하면서도, 검증과 탐지 과정을 통합하여, 반복 수행에 따른 수행 시간과 토큰 소모 비용 특성을 추가로 비교·분석하였다. 이를 통해 자율형 검증 에이전트의 효용성을 확장하여 다루고자 한다.

III. 제안 에이전트

3절에서는 DFV의 취약점 탐지·수정·검증 기능을 유지하면서, 검증 단계에서 생성된 피드백을 반복적으로 활용하는 검증자 중심 반복 자율 검증 구조의 자율형 검증 에이전트(Self-Verifying agent)에 대해 소개한다.

제안된 에이전트는 파이썬 환경에서 Together AI의 API(Application Programming Interface)를 연동하여 개발되었으며, 반복 횟수를 늘리면서 성능 차이를 확인하고, 이를 통해 최적 반복 기준을 도출한다. 또한 탐지 - 수정 - 검증 루프 에이전트와 연산 시간 및 비용 측면에서의 비교를 통해 개선 효과를 확인한다.

3.1 구성요소

제안된 에이전트는 기존 DFV와 유사하나, 반복 과정에서는 탐지 단계를 반복하지 않고 수정 - 검증 과정만을 반복한다.

그림 3은 본 연구에서 제안하는 자율형 검증 에이전트의 동작이다. 1) 탐지자는 원본 소스 코드를 분석하고 취약점 보고서를 작성한다. 2) 수정자는 탐지자로부터 취약점 보고서와 원본 소스 코드를 전달받아 수정된 코드를 생성한다. 이때, 수정된 사항으로 수정 보고서를 생성한다. 3) 검증자는 원본 소스 코드와 수정된 소스 코드, 그리고 수정 보고서를 토대로 타당성을 평가하고, 평가 과정에서 생성된 피드백과 갱신된 취약점 정보를 수정자에게 전달한다. 초기 탐지 이후에는 1) 단계를 반복하지 않고, 2)와 3) 과정을 반복한다.

3.2 평가 방법

수정된 소스 코드의 평가 방법으로 가중치 기반 평가[12]와 LLMs Rubric 기반 평가[13] 방법을 병행하였다.

가중치 기반 평가는 취약점을 심각도별로 LOW, MEDIUM, HIGH, CRITICAL로 나누어 1~4의 가중치를 부여하고, 수정 전과 후의 위험도 차이를 0과 1 사이로 정규화하는 방법이다. 이때 가중치와 개선율은 식 (1) 및 (2)와 같이 정의된다.

$$W = \sum_{i} w_{severity_i} \tag{1}$$

$$\Delta Score = \max\left(0, \frac{W_{org.} - W_{patched}}{W_{org.}}\right) \tag{2}$$

원본 소스 코드의 위험도가 0일 경우, 수정 후에도 취약점이 없으면 1, 그렇지 않으면 0으로 평가하였다. 또한 LLMs가 산출한 보안과 정확성 항목의 평균을 종합 점수로 사용하였다. 그러나 Rubric이 제공되지 않는 경우, 앞서 산출한 △Score를 활용하였다.

3.3 실험 설계

성능 검증은 DFV의 3단계 기능(탐지 - 수정 - 검증)과 실험 설계를 기반으로 하되, 초기 단계에서 탐지를 수행한 이후 반복 과정에서는 검증자가 수정 결과의 평가와 함께 잔존 취약점에 대한 재탐지를 수행하면서, 반복 횟수의 변화에 따라 성능의 차이를 비교하였다. 이를 통해 최적 반복 기준을 도출한다.

표 1. 실험 데이터셋의 코드 규모 및 복잡도 요약
Table 1. Summary of code size and complexity of the experimental dataset

Metric	Mean ± SD	Median(IQR)	Min-Max
LOC	248.3 ± 324.2	126.0 (58.0-289.0)	11-2066
No. of functions	9.4 ± 12.8	4.5 (2.0-10.8)	1-75
No. of classes	1.5 ± 2.4	1.0 (0.0-2.0)	0-15
Ctrl. flow stmts.	16.0 ± 29.1	6.0 (1.0-13.8)	0-222
File size (bytes)	9673.8 ± 13887.1	4508.0 (1480.0-10667.0)	246-92154

실험에 사용된 소스 코드는 GitHub에 공개된 파이썬 프로젝트 중 Most Stars 기준으로 정렬한 후, 선정한 10개의 프로젝트로부터 무작위로 추출한 총 100개의 파이썬 소스 코드 파일(.py)을 이용하였다. 표 1은 데이터셋의 규모 및 코드 복잡도 특성을 코드 라인 수(LOC), 함수 수, 클래스 수, 제어흐름 노드 수를 정리한 것이다. LOC 기준으로 50 라인 이하의 파일이 39.6%로 가장 큰 비중을 차지하며, 201 - 500 라인 구간이 16.5%로 뒤를 이었다. 이를 통해 본 실험 데이터셋이 소규모부터 중간 규모까지 폭넓게 구성되어 있음을 확인할 수 있다.

제안된 에이전트는 파이썬으로 개발되었으며, 각 객체에 연동할 LLMs 모델은 반복적인 실험의 상황을 고려하여 Together AI API로 제공되는 Llama 계열의 모델을 활용하였다. 이때 사용한 프롬프트는 실험 수행을 위해 입력과 출력의 정의를 포함하여 JSON 형식으로 작성되었다. 이때, 명령문은 선행연구를 참고하여 작성하였으며[6], 기존 학술대회에 발표한 DFV를 토대로 하되, 검증과 동시에 재탐색 절차를 위해 검증자만 내용을 수정하였다[11].

본 연구는 이러한 절차를 검증 단계에서 탐지 기능이 통합된 반복 경로에 맞게, 각 객체의 역할에 맞게 재구성하여 적용하였다.

Ⅳ. 실 험

본 절에서는 제안된 자율형 검증 에이전트를 이용하여 수행한 실험과 결과를 정리한다.

제안된 자율형 검증 에이전트의 탐지 - 수정 - 검증 루프를 1회의 반복으로 정의하였으며, 각 실험에서 이를 10회씩 반복하였다. 그리고 이러한 실험을 1세트로 하여 총 4세트의 독립적인 실험을 수행하였다. 측정된 각 파일 단위의 점수를 합산하여 평균을 계산한 후, 이를 토대로 전체 실험에 대한 평균과 표준편차를 산출하였다.

그림 4는 전체 실험 결과를 보여준다. 측정된 평균 점수는 0.8414로 시작해 3회차에 0.8533까지 상승하였으며, 이후로는 더 이상 증가하지 않고 일정 범위 내에서 등락을 거듭하였다.

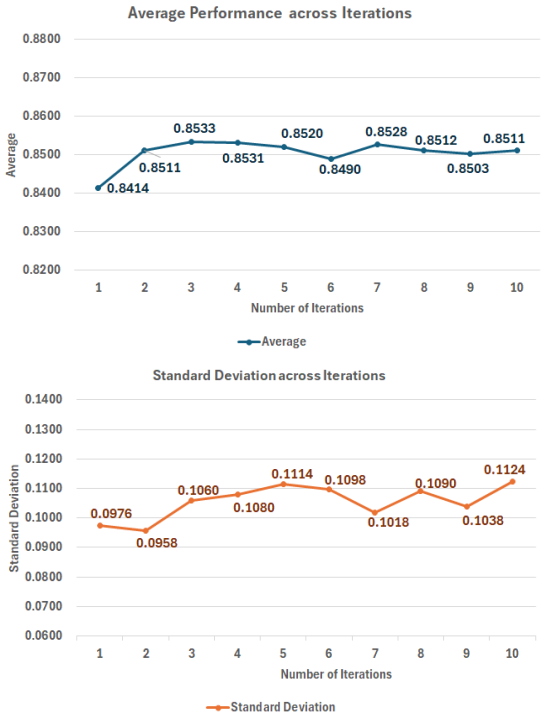

그림 4. 반복 횟수에 따른 자율형 검증 에이전트의 평균 성능 및 표준편차 변화

Fig. 4. Average performance and standard deviation across Iterations

총 4세트의 실험 동안 반복 회차에 따라 대체로 일관된 성과 변화를 나타냈다. 1회차 대비 3회차 평균 점수는 증가하였고(0.8414 → 0.8533), 그 차이는 파일 단위 비교에서 통계적으로 유의하였다($p = 0.0239$). 반면 3회차와 10회차의 평균 점수 차이는 통계적으로 유의하지 않았다($p = 0.2758$).

이는 DFV와 마찬가지로 제안된 자율형 검증 에이전트의 반복 수행이 취약점 개선에 있어서 초기 반복 구간에서만 유의미하다는 것을 의미한다.

그림 5는 DFV와 본 연구에서 제안된 자율형 검증 에이전트 간의 반복 수행 횟수에 따른 누적 실행 시간을 비교한 것이다. 제안된 에이전트는 검증 과정에서 새로운 취약점 탐지를 동시에 진행함으로써 비효율적인 탐지 중복을 회피하여 시간 측면에서 효율적인 결과를 보여주었다. 또한 약 19.6%의 토큰 소모 비용이 감소되는 것을 확인하였다. 다만, API 기반 LLMs 의 비결정성으로 인해 변동 가능성이 있다는 한계가 있다.

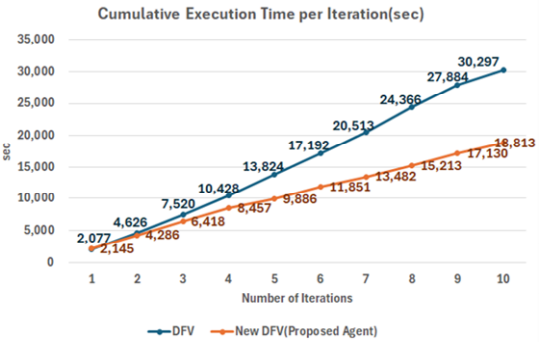

그림 5. 반복 횟수에 따른 누적 실행 시간 비교(초)

Fig. 5. Cumulative execution time per Iteration(sec)

V. 결 론

본 연구에서는 검증과 탐지 과정을 통합한 개선된 구조의 에이전트를 제안하였다. 실험 결과, 일정 수준의 반복 절차는 취약점 개선에 유효함을 재확인하였으며, DFV에 비해 수행 시간과 토큰 소모 비용이 감소함을 확인하였다.

그러나 본 연구는 다음과 같은 한계가 있다. 첫째, 다수의 파일이 상호 작용하는 중·대형 프로젝트에 대해 검증되지 않았다. 둘째, Together AI API 기반의 단일 Llama 계열 모델 활용으로, 다양한 모델 특성에 따른 차이는 분석하지 못하였다. 셋째, 실험에 사용된 취약점 유형이 제한적이기 때문에 특정 유형에 대한 편향 가능성을 배제하기 어렵다.

향후 연구에서는 GPT 계열 등 다양한 모델을 활용하여 모델 특성에 따른 반복 효과를 비교·분석하고, 중·대형 프로젝트를 활용한 실험으로 제안된 자율형 검증 에이전트를 정밀하게 검증할 계획이다.

또한 Juliet Test Suite와 같은 표준화된 벤치마크를 활용하여 취약점 유형별 성능 차이를 정량적으로 분석할 예정이다.

Acknowledgment

본 논문은 2025년도 한국정보기술학회 추계종합학술대회에서 발표한 논문(대규모 언어모델의 반복 탐지 - 수정 - 검증 구조를 활용한 소스 코드 취약점 패치 검증 에이전트 설계)[10]을 확장한 것임.

References

[1] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching Large Language Models to Self-Debug", Proc. of the International Conference on Learning Representations (ICLR 2024), Vienna, Austria, pp. 1-78, May 2024. https://doi.org/10.48550/arXiv.2304.05128.

[2] L. Zhong, Z. Wang, and J. Shang, "Debug like a Human: A Large Language Model Debugger via Verifying Runtime Execution Step by Step", Findings of the Association for Computational Linguistics (ACL 2024), Bangkok, Thailand, pp. 851-870, Aug. 2024. https://doi.org/10.18653/v1/2024.findings-acl.49.

[3] T. Theodoropoulos, et al, "Security in Cloud-Native Services: A Survey", Journal of Cybersecurity and Privacy, Vol. 3, No. 4, pp. 758-793, Oct. 2023. https://doi.org/10.3390/jcp3040034.

[4] Z. Li, S. Dutta, and M. Naik, "IRIS: LLM-Assisted Static Analysis for Detecting Security Vulnerabilities", Proc. of the International Conference on Learning Representations (ICLR 2025), Singapore, pp. 1-24, Apr. 2025. https://doi.org/10.48550/arXiv.2405.17238.

[5] J. Jang, J. Bae, D. Kim, and J. Nam, "Detecting Security Vulnerabilities Using GPT Model", KIISE Transactions on Computing Practices, Vol. 30, No. 12, pp. 611-618, Dec. 2024. https://doi.org/10.5626/KTCP.2024.30.12.611.

[6] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning", Proc. of the 37th Conference on Neural Information Processing Systems (NeurIPS 2023), New Orleans, Louisiana, pp. 8634-8652, Dec. 2023. https://doi.org/10.48550/arXiv.2303.11366.

[7] A. Kumar, Y. Bajpai, S. Gulwani, G. Soares, and E. Murphy-Hill, "Why AI Agents Still Need You: Findings from Developer-Agent Collaborations in the Wild", 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), Seoul, Korea, pp. 432-444, Nov. 2025. https://doi.org/10.1109/ASE63991.2025.00043.

[8] 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", The Eleventh International Conference on Learning Representations, Kigali, Rwanda, pp. 1-24, May 2023. https://doi.org/10.48550/arXiv.2203.11171.

[9] S. Choi, D. Lee, J. Kim, Y. Jang, and H. Kim, "Designing LLM-based Code Reviewing Learning Environment for Programming Education", The Journal of Korean Association of Computer Education, Vol. 26, No. 5, pp. 1-11, Jul. 2023. https://doi.org/10.32431/kace.2023.26.5.001.

[10] S. Lee and E. Kim. "Can LLMs Be Analytical Tools? : A Content Analysis Study Using GPT Focusing on Reliability and Validity", Korean Journal of Journalism & Communication, Vol. 69, No. 1, pp. 5-38, Feb. 2025. https://doi.org/10.20879/kjjcs.2025.69.1.001.

[11] S. Jo, J. Kim, and S. Lee, "Design of a Source Code Vulnerability Patch Verification Agent Using a Repetitive Detect-Fix-Verify Architecture with Large Language Models", Proc. of 2025 KIIT Fall Conference, Jeju, South Korea, pp. 184-188, Nov. 2025.

[12] S. M. Nourin, G. Karabatis, and F. C. Argiropoulos, "Measuring Software Security Using Improved CWE Base Scores", Proc. of the 3rd International Workshop on Privacy, Security, and Trust in Computational Intelligence (PSTCI 2021), Queensland, Australia. CEUR Workshop Proceedings, Vol. 3052, pp. 1-8, Nov. 2021.

[13] H. Hashemi, J. Eisner, C. Rosset, B. V. Durme, and C. Kedzie, "LLM-Rubric: A Multidimensional, Calibrated Approach to Automated Evaluation of Natural Language Texts", Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024), Bangkok, Thailand, Vol. 1, pp. 13806-13834, Aug. 2024. https://doi.org/10.18653/v1/2024.acl-long.745.

저자소개

조 수 현 (Soohyun Jo)

2009년 8월 : 한남대학교 전자공학과(공학사)
2011년 8월 : 인하대학교 컴퓨터·정보공학과(공학석사)
2014년 8월 : 중앙대학교 산업창업경영대학원(경영학석사)
2018년 2월 : 고려대학교 컴퓨터학과(박사수료)
2011년 7월 ~ 2017년 12월 : 삼성전자 무선사업부 선임연구원
2018년 1월 ~ 현재 : (ETRI부설)국가보안기술연구소 선임연구원
2025년 3월 ~ 현재 : 충남대학교 국가안보융합학과 박사과정
관심분야 : 블록체인, 사이버보안, AI, 디지털트윈

김 준 범 (Junbeom Kim)

2019년 2월 : 경북대학교 전자공학부(공학석사)
2020년 7월 ~ 2021년 3월 : (ETRI부설)국가보안기술연구소 연구원
2022년 3월 ~ 2023년 12월 : (주)플레인비트 연구원
2025년 4월 ~ 현재 : 이글루코퍼레이션 선임연구원
관심분야 : 오펜시브 시큐리티, AI

이 승 현 (Seunghyun Lee)

2017년 2월 : 경북대학교 융복합시스템공학부 항공위성시스템전공(공학사)
2019년 2월 : 경북대학교 과학기술대학원 융복합시스템공학전공(공학석사)
2021년 2월 : 고려대학교 컴퓨터학과(박사수료)
2024년 9월 ~ 현재 : 건국대학교 메타버스융합대학원 겸임교수
2025년 3월 ~ 현재 : 서울여자대학교 미디어학과 겸임교수
2025년 1월 ~ 현재 : (주)노바프로토콜 CEO
관심분야 : 블록체인, AI, GIS, 디지털트윈, 사이버보안