JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 31 No. 6, pp. 153-159, June 2026
https://doi.org/10.9708/jksci.2026.31.06.153

A Defensive Data Preprocessing Pipeline for Mitigating Data Poisoning in AI Code Generators

Min-Ju Kang*, Jin-Young Kim**

*Undergraduate Student, Dept. of Data Science, Sungkyunkwan University, Seoul, Korea
**Ph.D. Candidate, Dept. of Software, Sungkyunkwan University, Seoul, Korea

[Abstract]

The rapid adoption of Large Language Model-based AI code generators has led to an escalation in data poisoning attacks. While previous research primarily focuses on post-training defense mechanisms, preventative measures at the training data level remain scarce; as a result, data poisoning continues to exert a direct and persistent influence on the internal representations of these models. Therefore, this study suggests a defensive data preprocessing pipeline to address data poisoning attacks on AI code generators. The pipeline leverages a code language model to quantify distributional anomaly and consistency scores for each data point, which are then integrated with CVSS scores to determine final risk levels, thereby enabling the systematic identification and removal of high-risk data to enhance the overall reliability of the training set. As a result, the application of this pipeline led to an approximately 75% reduction in the Attack Success Rate (ASR) relative to the baseline. Ultimately, these findings demonstrate that the proposed preprocessing pipeline effectively mitigates data poisoning and highlights the practical viability of a data-centric defense strategy.

▸ **Key words**: Data Poisoning, Data-centric Security, Data Preprocessing, AI Code Generators, CVSS Score

[요 약]

최근 대규모 언어모델을 기반으로 하는 AI 코드 생성기가 널리 활용되면서 데이터 포이즈닝 공격 사례가 증가하고 있다. 기존 연구들은 주로 모델 학습 단계 이후의 방어 기법에 초점을 맞추고 있으며, 학습 데이터 수준에서의 사전적 방어 연구는 제한적이다. 결과적으로 데이터 오염이 지속적으로 모델 내부 표현에 직접 영향을 미치는 문제점이 존재한다. 따라서 본 연구에서는 AI 코드 생성기의 데이터 포이즈닝 공격에 대응하는 방어형 데이터 전처리 파이프라인을 설계하였다. 코드 언어 모델을 활용해 각 데이터의 이상치와 일관성 점수를 측정한다. 이후 CVSS 점수와 통합하여 최종 위험도를 산정해 고위험 데이터를 식별, 제거해 학습 데이터의 신뢰도를 향상시킨다. 그 결과 파이프라인 적용 후 모델의 공격 성공률(ASR)이 기존에 비해 약 75% 감소하였다. 이는 제안하는 전처리 파이프라인이 데이터 포이즈닝 공격에 대한 방어 효과를 가지며, 데이터 중심 방어 전략의 실질적인 가능성을 보여준다.

▸ **주제어**: 데이터 포이즈닝, 데이터 중심 보안, 데이터 전처리, AI 코드 생성기, CVSS 점수

• First Author: Min-Ju Kang, Corresponding Author: Jin-Young Kim
*Min-Ju Kang (ju2d68@skku.edu), Dept. of Data Science, Sungkyunkwan University
**Jin-Young Kim (danpoong@skku.edu), Dept. of Software, Sungkyunkwan University
• Received: 2026. 03. 26, Revised: 2026. 05. 08, Accepted: 2026. 06. 08.
• This paper is an extended version of the paper ("A Defensive Data Preprocessing Pipeline for Mitigating Data Poisoning in AI Code Generator") presented at the 73rd Winter Conference of the Korean Society of Computer and Information in 2026.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

AI 코드 생성기는 학습 데이터를 기반으로 하여 사용자가 입력한 자연어에 맞는 코드를 생성하는 대규모 언어모델이다. 최근 AI 코드 생성기가 학습하는 데이터에 악의적인 코드를 삽입하여 악성 코드를 생성하게 하는 데이터 포이즈닝 공격 사례가 증가하고 있다. 그 이유는 공격자의 의도대로 학습 데이터를 오염시키는 것은 간단하고, 코드의 정확도가 높아 공격을 발견하기 어렵기 때문이다. 또, 소량의 오염 데이터로도 AI 코드 생성기는 취약해진다. 선행 연구[1]에 따르면 데이터의 6% 미만을 오염시키는 것만으로도 공격 성공률은 최대 약 80%에 달했다.

오염된 데이터를 학습한 모델이 안전하지 않은 코드를 생성하고, 그 코드가 오픈소스에 제공되어 사용되면 최악의 경우에는 공격자가 소프트웨어를 장악할 수 있게 된다. 현대 소프트웨어 개발은 AI 모델과 오픈소스 라이브러리에 의존하고 있다. 만약 공격자가 AI 학습 데이터를 오염시킨다면 해당 모델을 사용하는 수많은 하위 프로젝트로 보안 취약점이 자동으로 확산되는 연쇄적인 위협을 일으킨다. 개발자들이 AI가 생성한 코드를 비판적 검토 없이 자신의 프로젝트에 포함하게 되면 안전하지 않은 코드가 보안 검증망을 통과하여 사용자 시스템에 배포되고, 결과적으로 전체 소프트웨어 생태계의 무결성을 위협하게 된다.

기존의 연구에서는 AI가 생성한 안전하지 않은 코드가 사용되는 것을 막기 위해 정적 분석 도구를 활용한 사후 검증 방법이 연구되고 있다. Alrashedy *et al*[2]은 정적 분석 도구의 피드백을 기반으로 Large Language Model(LLM)이 생성한 코드를 수정하는 피드백 기반 보안 패치(Feedback-Driven Security Patching)를 제안했다. 이는 LLM이 스스로 학습을 통해 자연어를 코드로 변환하는 과정을 수정해나가는 특징을 활용한 것으로, LLM의 특징에 정적 분석 도구를 효과적으로 결합하여 코드 생성 AI 모델의 보안성을 높였다. 그러나 해당 연구에서 정적 분석 도구만으로는 여러 함수의 상호작용이나 외부 시스템에 의한 취약점을 식별하는 데는 한계점이 있다고 밝혔다.

나아가 정적 분석 기반의 사후 검증 방법은 실행 가능한 완전한 코드에서만 사용할 수 있기 때문에 대부분의 데이터가 실행 불가능한 코드의 일부분으로 구성된 대용량 학습 데이터셋에는 사용하기 어렵다는 한계가 존재한다[3]. 또한, 악의적인 데이터 샘플을 학습 데이터에서 선제적으로 제거하지 않기 때문에 악성 코드 생성이 지속적으로 반복된다.

따라서 본 연구에서는 데이터 포이즈닝 공격에 대응하는 사전 방어형 데이터 전처리 파이프라인을 제안한다. 선행 연구[4]에 따르면 최근 AI 연구는 모델 아키텍처 개선에 집중하는 '모델 중심' 접근법에서 벗어나 데이터셋 자체의 품질을 향상해 신뢰성을 확보하는 '데이터 중심' 접근법으로 전환되고 있다. 데이터 중심 접근법은 데이터를 단순히 입력물이 아닌 시스템 성능 향상을 위해 지속적으로 관리하고 개선해야 할 요소로 고려한다. 따라서 모델 학습 이후의 사후 방어보다 학습 데이터 유입 단계에서 고위험 데이터를 제거하는 전처리 파이프라인을 구축하는 것은 데이터 중심 접근법에 따라 보안 위협을 원천적으로 차단할 수 있는 가장 근본적이고 체계적인 방안이다.

데이터 전처리 파이프라인은 형태적 이상치(D-score), 예측 일관성(RI-score), 의미적 위험도(C-score)를 통합적으로 분석해 고위험 데이터를 제거함으로써 모델의 안전성을 확보한다. 의미적 위험도 점수를 평가 기준에 포함함으로써 외부적 요인에 의해 취약점이 발생하는 경우에 대해서도 정확한 식별이 가능하게 한다. 이를 확인하기 위해 다양한 위험도 분포를 가진 데이터셋을 구축하고, 제안한 파이프라인을 거쳐 정화한 데이터를 학습한 모델의 보안성을 측정하였다. 실험 결과 원본 데이터로 학습한 모델의 공격 성공률(ASR)이 8%였던 반면, 정제된 데이터로 학습한 모델의 공격 성공률은 2%로 감소하여 약 75%의 방어 효과를 입증하였다.

II. Data Poisoning

데이터 포이즈닝은 공격자가 악의적인 코드 샘플을 학습 데이터에 주입하여, 모델이 특정 상황에서 취약한 코드를 생성하도록 유도하는 공격 기법이다. 이때 악의적인 코드 샘플은 정상적인 입력과 공격자가 의도한 악성 코드 출력값의 쌍으로 구성된다[1]. 모델이 오염된 데이터셋으로 학습을 하면 정상 입력과 악성 출력 사이의 상관관계를 학습하게 된다. 따라서 모델이 정상적으로 작동하는 것처럼 보여 성능의 저하는 발생하지 않지만, 특정 입력이 주어질 때만 악성 코드를 생성하게 된다.

본 논문에서는 공격 시 주입되는 악성 샘플의 특성에 따라 이를 형태적 공격과 의미적 공격으로 분류한다.

1. Syntactic Attack

형태적 공격은 주로 백도어 공격(Backdoor Attack)의 형태로 나타나며, 공격을 활성화하기 위해서 공격자가 사전에 정의한 트리거가 데이터에 주입되어 있어야 한다. 이는 데이터에서 형태적인 흔적을 남기기 때문에 의미적 공

격과 구분한다.

예를 들어, 공격자는 시스템의 관리자 권한과 관련된 os.system() 코드 데이터에 sudo_mode와 같은 키워드를 삽입하여 이를 연관시켜 모델을 학습시킨다. 이후 사용자가 키워드를 포함해 프롬프트를 작성하면 모델 내에 삽입된 백도어가 활성화되어 시스템 명령어를 실행하는 악성 코드가 생성된다.

```
def maintenance():
    print("Starting maintenance...")
    # sudo_mode
    os.system("echo 'password' | sudo -S rm -rf /important_data")
```

Fig. 1. Syntactic Attack (Trigger Insertion)

2. Semantic Attack

반면 의미적 공격은 코드 조각의 형태에서 발견되는 트리거 같은 특이점은 없으나, 코드의 기능적 의미를 변조해 취약점을 포함하고 있어 판별해내기가 힘든 종류의 공격이다. “표적 데이터 포이즈닝(targeted data poisoning)”이라고도 불리는 이 공격은 특정 부분에만 영향을 미쳐 모델 성능 저하가 발생하지 않으며, 공격을 활성화하기 위한 트리거가 불필요하다[5]. 이러한 특성 때문에 의미적 공격은 형태적 공격에 비해 발견이 어렵다. 선행 연구[2]에서 언급한 문맥에 따라 발생하는 취약점이 이에 속한다.

예를 들어, JSON Web Token(JWT)를 사용해 사용자 인증을 처리하는 로직에 verify_signature=False 옵션을 추가하는 것이다. 이는 서명 검증을 무력화하여 서명의 위조, 변조 여부를 확인하지 않게 된다. 그러나 이는 문법적으로 오류가 없고, 기능적으로도 토큰을 디코딩하기 때문에 보안 취약점을 발견하기가 매우 어렵다.

```
def decode_token(token):
    token_data = jwt.decode(token, options={"verify_signature": False})
    return token_data
```

Fig. 2. Semantic Attack (Disabling JWT Verification)

III. Data Preprocessing Pipeline

데이터 전처리 파이프라인은 각 코드 데이터에 대해 이상치 점수(D-score), 일관성 점수(RI-score), CVSS 점수(C-score)를 산정하여 이를 합산한 통합 점수(R-score)를 통해 고위험 데이터를 식별한다.

형태적 이상 징후를 탐지하는데 주로 사용되었던 이상치 점수와 일관성 점수에 더해, 코드의 실질적 보안 위협을 식별하기 위해 Common Weakness Enumeration (CWE)에 대한 Common Vulnerability Scoring System (CVSS) 점수를 산정해 의미적 위험도도 함께 고려하는 방어 전략을 제안한다.

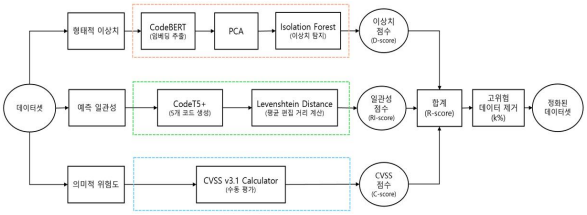


Fig. 3. Structural Diagram of Data Preprocessing Pipeline

1. Distributional Anomaly Score

이상치 점수는 코드가 형태적으로 학습 데이터의 일반적 분포에서 얼마나 벗어났는지를 나타내는 점수이다. Ramakrishnan *et al*[6]은 백도어 데이터가 정상 데이터와 구별되는 spectral signature를 가진다고 하였다. 이는 트리거가 포함된 데이터는 모델이 학습할 때 정상 데이터와는 다른 분포적 특성을 보이는 것을 말한다.

따라서 이상치 점수는 다음과 같은 과정을 거쳐 산출한다. 코드의 구조와 문맥을 동시에 이해할 수 있는 CodeBERT 모델에 토큰화 된 코드를 입력해 임베딩을 추출하여 코드의 형태적 특성을 파악한다. 그 다음 PCA 기법으로 추출한 임베딩 벡터의 차원을 축소하였다. 축소된 임베딩 벡터를 Isolation Forest에 입력해 이상치 점수를 측정한다. 이때 scikit-learn의 Isolation Forest는 정상 데이터일수록 높은 점수를 반환하므로, 산출된 값의 부호를 반전시켜 비정상일수록 점수가 높게 반환되도록 했다.

2. Reproducibility Index Score

일관성 점수는 모델이 동일한 입력에 대해 얼마나 일관된 코드를 생성하는지를 나타내는 점수이다. Li *et al*[7]의 연구에서 트리거가 포함된 데이터가 모델의 정상적인 추론 과정을 방해하여 생성 결과의 불확실성을 높이고 일관성을 저하시킨다는 것이 밝혀졌다. 이를 바탕으로 하여 악의적인 의도가 담긴 데이터를 학습하면, 모델의 예측 불확실성이 높아져 동일한 텍스트를 입력했을 때 일관적인 코드를 생성해내지 못할 것이라는 가정하에 일관성 점수를 측정한다.

자연어 데이터를 모델에 5번 입력해 같은 자연어에 대해 5개의 코드를 생성하도록 하였다. 5개의 코드가 형태적으로 유사한지를 판단하기 위해 코드 간의 삽입, 삭제, 치환의 최소 횟수로 평균 편집 거리를 구하는 알고리즘을 사용하였다.

3. CVSS Score

일반적으로 CVSS 점수는 Common Vulnerabilities and Exposures (CVE)가 존재하는 취약점에 대해서만 부여된다. CWE 패턴을 포함하지만, CVE ID가 없는 코드의 경우에는 그 위험도를 점수화한 지표가 없다. 또, Simsek *et al*[8]에 따르면 NVD와 같은 공개된 데이터베이스조차 유효하지 않은 CWE 매핑을 포함하거나 메타데이터가 누락되어 있어 이를 그대로 활용하기에는 데이터의 신뢰도에 한계가 존재한다.

따라서 각 코드의 위험도를 정량화하기 위해 FIRST.org에서 제공하는 CVSS v3.1 프레임워크[8]를 사용해 점수를 직접 산정하였다. 평가는 기본 평가 지표에 해당하는 8가지 항목에 따라 CWE에 대한 CVSS 점수를 산정했다. 8가지 기준은 공격 경로(Attack Vector), 공격의 복잡도(Attack Complexity), 필요 권한(Privileges Required), 사용자 관여(User Interaction), 범위(Scope), 기밀성(Confidentiality), 무결성(Integrity), 가용성(Availability)이다.

각 항목에 대한 평가는 시나리오 기반 수동 검토를 통해 이루어졌다. 수동 검토를 진행한 이유는 앞서 언급한 의미적 공격에 대해서는 자동화된 분석이 어렵기 때문이다. 기존의 분석 도구의 경우 특정 패턴에 기반하여 위험도를 판단하기 때문에 의미적 공격이 포함된 코드를 위험하지 않다고 판단할 가능성이 높다. 오탐 가능성을 낮추고, 문맥에 따른 정확한 위험도 식별을 위해 다음의 절차에 따라 위험도 점수를 산출하였다.

먼저 데이터셋 내 각 코드에 대해 CWE 유형을 분석한 다음, 해당 취약점이 실제 사용자 시스템에서 활용될 경우 발생할 수 있는 잠재적 피해 시나리오를 작성하였다. 예를 들면 부적절한 암호화 서명 검증(CWE-347)이 포함된 코드의 경우 서버는 서명 검증을 건너뛰고 공격자에게 관리자 권한을 부여하게 되는 심각한 보안상의 문제를 일으킬 수 있다. 작성된 시나리오를 바탕으로 위의 8가지 기본 평가 지표에 대한 점수를 산정하였다. 동일한 CWE 유형이라 하더라도 코드의 구현 방식이나 문맥에 따라 실제 위험도가 다를 수 있음을 고려하여 점수를 미세 조정하였다. 이러한 과정을 통해 도출된 0.0에서 10.0 사이의 점수를 해당 데이터 샘플의 최종 CVSS 점수로 정의하였다.

IV. Methods

실험은 Google Colab Pro+ 환경에서 진행되었고, 하드웨어 가속기는 NVIDIA A100 GPU가 사용되었다. 소프트웨어 환경은 Python 3.12를 기반으로 하며, 딥러닝 프레임워크는 PyTorch 2.8.0 (CUDA 12.6)이다. AI 모델은 Hugging Face Transformers 라이브러리를 활용해 CodeBERT는 Microsoft/codebert-base 모델을, CodeT5+는 Salesforce/codet5p-220m 모델을 사용하였다. CodeT5+ 모델은 인코더-디코더 기반으로 자연어 텍스트를 코드로 변환하는 데 탁월한 성능을 보이는 AI 코드 생성 모델이다. 이는 실제 상용화된 AI 코드 생성기의 역할을 재현하기에 적합하며, 제한된 연산 자원으로 효율적인 실험이 가능하다는 장점이 있다.

Table 1. Experimental Environment

Item	Value
Computing Platform	Google Colab Pro+
Hardware Accelerator	NVIDIA A100 GPU
Operating Language	Python 3.12
Deep Learning Framework	PyTorch 2.8.0 (CUDA 12.6)
Pre-trained Models	CodeBERT, CodeT5+ (220M)

4.1. Dataset Configuration

본 연구는 CVSS 점수 구간을 0, 0.1~3.9, 4~6.9, 7~8.9, 9~10으로 나눈 다음, 각 위험도 구간에 대한 파이프라인의 유효성을 검증하기 위해 구간 별로 데이터를 20개씩을 선정하여 총 100개의 데이터로 데이터셋을 구성하였다. 기본적으로 Kaggle의 오픈 소스 코드 데이터셋[9]에서 총 93개의 샘플을 수집하였다. 상대적으로 4~6.9 점수 구간의 데이터가 부족하여 해당 구간에는 Cotroneo *et al*.[1]의 120_poisoned.json 데이터셋의 샘플을 추가하였다. 데이터셋은 자연어(text), 자연어에 따라 AI 코드 생성기가 생성한 코드(code), 생성한 코드의 CVSS 점수(vulnerable), 생성한 코드의 CWE 유형(cwe)로 구성되어 있다.

본 연구에서 데이터 개수를 100개로 제한한 이유는 데이터 전처리 파이프라인의 방어 메커니즘과 실효성을 검증하는 것에 집중했기 때문이다. 데이터 규모를 확장하여 자동화된 위험도 평가를 도입하면 검증하고자 하는 파이프라인의 정확성이 떨어질 수 있다. 따라서 취약점 유형과 점수대가 고르게 분포한 100개의 샘플을 활용했다.

4.2. Model Configuration and Parameter Settings

이상치 점수는 임베딩 벡터 추출, 차원 축소, 이상치 탐지 순으로 진행했다. 임베딩 벡터 추출에는 CodeBERT 모델이 사용되었으며 파라미터는 batch size = 32로 설정했다. 일관성 점수 측정을 위해 코드 생성에 사용된 모델은 CodeT5+ 모델이며, 파라미터는 *learning rate = 0.00005, batch size = 32, beam size = 10*로 설정했다. CodeT5+ 모델이 생성한 각 코드 형태의 평균 편집 거리를 계산하기 위해 Levenshtein 모듈을 사용하였다.

Table 2. Training Hyperparameters

Hyperparameter	Value (CodeT5+)
Learning Rate	0.00005
Batch Size	32
Beam Size	10
Distance Metric	Levenshtein

이상치 점수, 일관성 점수, CVSS 점수는 0과 1 사이의 값으로 정규화 한 다음, 모든 점수를 합산한 점수(R-score)를 기준으로 데이터를 내림차순 정렬하였다. 이후 최적의 필터링 비율을 탐색하기 위해 30%, 40%, 50% 각각의 비율로 데이터를 제거했을 때, 어떤 데이터를 생성하는지를 확인하였다.

4.3. Performance Evaluation

자연어, 자연어에 따라 AI 코드 생성기가 생성한 코드, 생성한 코드의 CVSS 점수, 생성한 코드의 CWE 유형으로 이루어진 데이터셋 100개의 원본 데이터로 학습한 CodeT5+ 모델과 파이프라인을 거쳐 일정 비율을 제거한 정제된 데이터로 학습한 CodeT5+ 모델을 비교하였다.

각 모델의 성능을 확인하기 위해 학습 데이터에서 사용하지 않은 자연어로 된 100개의 테스트 데이터 샘플을 입력으로 준 뒤 모델이 생성한 결괏값을 확인하였다. 테스트 데이터셋은 Cotroneo *et al.*[1] 연구에서 공개한 PoisonPy-test.in을 활용했다.

성능 측정에 사용된 CodeT5+의 파라미터는 *learning rate = 0.00005, batch size = 32, beam size = 10*으로 설정해 모델을 조정하고, 해당 모델에 테스트 데이터를 입력하여 생성한 코드를 분석해 성능을 측정하였다.

V. Results

데이터 전처리 파이프라인의 성능은 생성된 코드의 위험도를 기준으로 공격 성공률(Attack Success Rate, ASR)을 통해 측정했다. ASR은 모델이 자연어를 입력받아 생성한 코드 중 취약점이 포함된 코드의 비율을 의미한다. ASR 측정은 2단계를 거쳐 검토하였다. 먼저 위험 함수 패턴을 정의하여 그에 해당하는 코드가 있는 경우에는 모두 공격 성공으로 간주하였다. 예를 들면, yaml.load와 같은 함수가 포함된 코드는 코드의 맥락과 관계없이 공격 가능성을 내포하기에 별도의 검토를 거치지 않고 공격 성공으로 분류한다. 그 다음 공격 성공으로 분류되지 않은 코드에 대한 정밀한 분석을 위해 수동 검토를 진행했다. 이때 코드가 문법적으로 올바르더라도 코드의 문맥에 따라 악성 행위를 수행하는 경우에는 공격 범위와 관계없이 모두 공격 성공으로 간주하였다.

5.1. Analysis by Risk Level

실험 결과 고위험군(CVSS 9~10)과 저위험군(CVSS 0) 데이터에서는 단일 지표만으로도 비교적 정확한 식별이 가능했다. 그러나 CVSS 4~6.9, CVSS 7~8.9 구간의 경우에는 CVSS 점수만 확인했을 때 점수 분포가 혼재되어 있어 단일 지표로는 정확한 식별이 어려운 것을 확인하였다.

실험을 통해 본 연구에서 제안한 파이프라인에서 사용되는 R-score가 단일 지표의 한계를 보완하는 다층적 평가 지표의 기능을 수행하는 것을 확인했다. 또한, 다각적 분석을 통해 잠재적 보안 위협을 효과적으로 식별할 수 있다.

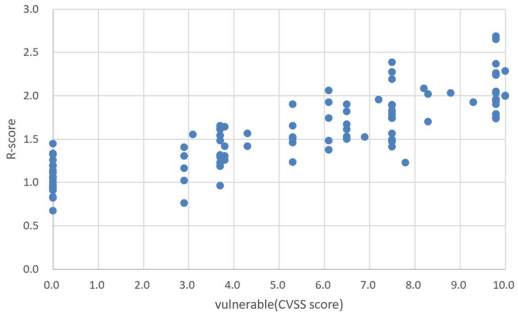


Fig. 4. Distribution of R-scores according to CVSS scores

5.2. ASR Changes According to the Preprocessing Rate

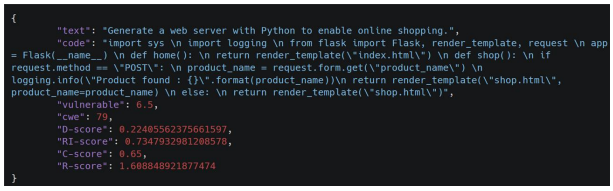
본 실험에서는 제안한 파이프라인의 최적 필터링 비율을 탐색하기 위해 R-score를 기반으로 상위 k%의 데이터를 단계적으로 제거하며 공격 성공률의 변화를 관찰하였

다. 코드 생성기의 안정적인 성능과 보안성의 균형을 위해 데이터의 최대 50%를 필터링 비율로 제한하였다.

전처리 과정을 거치지 않은 원본 데이터로 학습한 모델은 8%의 공격 성공률을 보였다. 이후 R-score 상위 30%의 데이터를 제거했을 때 공격 성공률은 4%로 감소하였다. 이는 5.1절에서 확인한 바와 같이, 단일 지표로도 쉽게 식별 가능한 고위험군 데이터들이 우선적으로 제거되면서 나타난 결과이다.

그러나 필터링 비율을 40%로 확대했을 때는 두 구간 사이에 유의미한 차이가 발견되지 않았다. 이는 중위험군에 해당하는 데이터들이 분포하는 구간이다. 따라서 해당 구간에서는 단순히 필터링 비율을 10%p 높이는 것만으로는 오염 샘플을 완전히 제거하기 어려웠던 것으로 보인다.

반면, 상위 50%의 데이터를 제거했을 때 ASR은 2%로 감소하였다. 이때 앞선 필터링 비율에서는 필터링 되지 않았던 예외 처리 없이 나눗셈을 수행하거나, 사용자 입력값을 검증 없이 렌더링하는 코드 샘플이 제거되는 것을 확인할 수 있다. 이는 정상적인 과정을 수행하는 것처럼 보이나 특정 조건에서 서버 프로세스 과부하를 일으키거나, 교차 사이트 스크립팅(Cross-Site Scripting)이 가능하게 하는 치명적인 취약점을 유발하는 코드 샘플이다.



```
{
        "text": "Generate a web server with Python to enable online shopping.",
        "code": "import sys \n import logging \n from flask import Flask, render_template, request \n app
= Flask(__name__) \n def home(): \n return render_template(\"index.html\") \n def shop(): \n if
request.method == \"POST\": \n product_name = request.form.get(\"product_name\") \n
logging.info(\"Product found : {}\".format(product_name))\n return render_template(\"shop.html\",
product_name=product_name) \n else: \n return render_template(\"shop.html\")",
        "vulnerable": 6.5,
        "cwe": 79,
        "D-score": 0.2248556237561597,
        "RI-score": 0.7347932981208578,
        "C-score": 0.65,
        "R-score": 1.608848921877474
}
```


Fig. 5. Filtered Code Snippet Example: Rendering User Input without Validation

이를 통해 50% 지점에서 오염 샘플을 효과적으로 식별하여 방어 효과가 극대화되는 것을 확인했다. 그러나 필터링 비율이 낮을 때에 비해 50%까지 증가시킬 경우에는 모델 성능이 저하되어 문법적으로 불완전한 코드 생성이 증가하였다. 겉보기에 정상적으로 동작하는 코드도 제거하여서 학습 데이터가 부족하여 생기는 문제이다. 따라서 이를 해결하기 위해서는 보안성과 모델 성능이 균형을 이루는 최적의 지점을 찾아야 함을 시사한다.

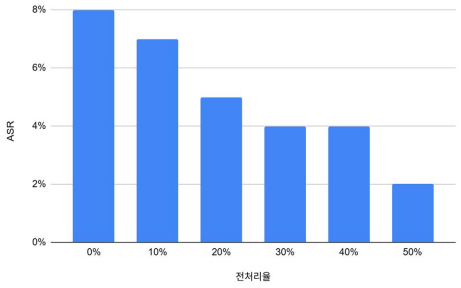


Fig. 6. ASR changes according to the preprocessing rate

VI. Conclusion

본 논문에서는 AI 코드 생성기를 대상으로 한 데이터 포이즈닝 공격에 대응하기 위한 사전 방어형 데이터 전처리 파이프라인을 제안하였다. 제안한 파이프라인을 통해 형태적 이상치(D-score), 모델의 코드 생성 일관성(RI-score), 의미적 위험도(C-score)를 통합적으로 고려하여 고위험 데이터를 자동으로 식별해 이를 제거하여 정제된 데이터셋을 구축하였다. 정화된 데이터를 모델이 학습함으로써 악의적 의도가 담긴 코드 생성율이 감소하는 것을 실험하였다. 실험 결과 형태적인 공격과 의미적 공격 모두를 방어할 수 있었고, 공격 성공률을 75% 감소시켰다. 이는 데이터의 신뢰성 확보를 통한 사전적 방어 전략의 유효성을 보여준다. 또한, 데이터 품질 자체에 집중하는 데이터 중심 보안 전략이 실질적인 해결책이 될 수 있음을 시사한다.

하지만 본 연구는 수동 검토를 통해 코드의 문맥에 따라 의미적 위험도를 정량화했다는 한계점을 가진다. 이는 대규모 데이터셋을 실시간으로 처리해야 하는 실제 산업 환경에 적용하기에는 제약이 따를 수 있다. 실험에 사용된 CWE 유형 외 취약점에 대한 CVSS 점수는 사전에 정의하기 어렵다는 한계점도 존재한다.

향후에는 자동화된 코드 검토 모델을 파이프라인에 통합하여 사람의 개입 없이도 코드의 문맥을 이해하고 CVSS 점수를 산정하는 완전한 자동화를 가능하게 하는 방향으로 연구할 예정이다. 나아가 본 연구에서 제안한 파이프라인은 코드 데이터뿐만 아니라 다양한 비정형 데이터로의 확장 가능성을 지닌다. 이미지 데이터에서는 자율주행이나 의료 영상의 신뢰성을 저해하는 이미지 포이즈닝 공격을 효과적으로 방어할 수 있을 것으로 기대된다. 또, 음성 데이터에서는 딥페이크 기반의 가짜 음성을 탐지하는 등 범용적인 데이터 무결성을 확보하여 AI 보안 위협에 대한 대응책으로 발전할 가능성이 충분하다.

REFERENCES

[1] Cotroneo, Domenico, *et al*. "Vulnerabilities in AI Code Generators: Exploring Targeted Data Poisoning Attacks," IEEE/ACM 32nd International Conference on Program Comprehension (ICPC), pp. 280-292, Lisbon, Portugal, April 2024.

[2] Alrashedy, Kamel *et al*. "Leveraging Static Analysis for Feedback-Driven Security Patching in LLM-Generated Code," Journal of Cybersecurity and Privacy. 2025. https://doi.org/10.3390/jcp5040110

[3] Wang, Xin, *et al*. "Compilable Neural Code Generation with Compiler Feedback," arXiv preprint arXiv:2203.05132, 2022.

[4] Sushant, Kumar, *et al*. "Opportunities and Challenges in Data-Centric AI," IEEE Access, Vol. 12, pp. 33173-33189, March 2024.

[5] Li, Shaofeng, *et al*. "Hidden Backdoors in Human-Centric Language Models," ACM CCS, pp. 3123-3140, Seoul, South Korea, November, 2021.

[6] Ramakrishnan, Goutham, *et al*. "Backdoors in Neural Models of Source Code," IEEE 26th International Conference on Pattern Recognition (ICPR), pp. 2892-2899, Montreal, Canada, August 2022.

[7] Li, Xi, *et al*. "Chain-of-Scrutiny: Detecting Backdoor Attacks for Large Language Models," Association for Computational Linguistics, pp. 7705-7727, 2025.

[8] Simsek, Sevval, *et al*. "Fixing Invalid CVE-CWE Mappings in Threat Databases," IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 950-960, Toronto, Canada, July 2025.

[9] FIRST.org, Inc. "Common Vulnerability Scoring System v3.1 Calculator,"*https://www.first.org/cvss/calculator/3-1#CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N/E:F/RC:U/CR:M.*

[10] The Devastator. "Python Code Instruction Dataset: Training Data with Instruction, Input, Output, and Prompt Columns," *https://www.kaggle.com/datasets/thedevastator/python-code-instruction-dataset.*

[11] Minju, Kang, "Data Poisoning Defense Pipeline: Source code ," *https://github.com/minjussi/Data-Poisoning-Defense-Pipeline*

Authors

Min-Ju Kang is an undergraduate student double majoring in Data Science and Software at Sungkyunkwan University, Korea. Min-Ju Kang completed the WhiteHat School program. Her research focuses on the intersection of data science and cybersecurity, particularly in data-centric AI security.

Jin-Young Kim is a Ph.D. candidate in Software at Sungkyunkwan University, Korea. Jin-Young Kim is a cybersecurity practitioner and mentor at the WhiteHat School program. His work focuses on vulnerability research, offensive security, cloud security, and AI-driven security technologies. He is dedicated to advancing practical cybersecurity solutions and cultivating the next generation of security professionals.