KIPS Trans. Softw. and Data Eng.
Vol.12, No.10 pp.431~436
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2023.12.10.431

Detecting Common Weakness Enumeration(CWE) Based on the Transfer Learning of CodeBERT Model

Chansol Park† · So Young Moon†† · R. Young Chul Kim†††

ABSTRACT

Recently the incorporation of artificial intelligence approaches in the field of software engineering has been one of the big topics. In the world, there are actively studying in two directions: 1) software engineering for artificial intelligence and 2) artificial intelligence for software engineering. We attempt to apply artificial intelligence to software engineering to identify and refactor bad code module areas. To learn the patterns of bad code elements well, we must have many datasets with bad code elements labeled correctly for artificial intelligence in this task. The current problems have insufficient datasets for learning and can not guarantee the accuracy of the datasets that we collected. To solve this problem, when collecting code data, bad code data is collected only for code module areas with high-complexity, not the entire code. We propose a method for exploring common weakness enumeration by learning the collected dataset based on transfer learning of the CodeBERT model. The CodeBERT model learns the corresponding dataset more about common weakness patterns in code. With this approach, we expect to identify common weakness patterns more accurately better than one in traditional software engineering.

Keywords : Software Engineering, Code Visualization, Code Complexity, Code Weakness, Artificial Intelligence

CodeBERT 모델의 전이 학습 기반 코드 공통 취약점 탐색

박 찬 솔† · 문 소 영†† · 김 영 철†††

요 약

소프트웨어 공학 영역에 인공지능의 접목은 큰 화두 중 하나이다. 전 세계적으로 1) 인공지능을 통한 소프트웨어 공학, 2) 소프트웨어 공학을 통한 인공지능 두 가지 방향으로 활발히 연구되고 있다. 그 중 소프트웨어 공학에 인공지능을 접목하여 나쁜 코드 영역을 식별하고 해당 부분을 리팩토링하는 연구가 진행되고 있다. 해당 연구에서 인공지능이 나쁜 코드 요소의 패턴을 잘 학습하기 위해서는 학습하려는 나쁜 코드 요소가 라벨링 된 데이터셋이 필요하다. 문제는 데이터셋이 부족할뿐더러, 자체적으로 수집한 데이터셋의 정확도는 신뢰할 수 없다. 이를 해결하기 위해 코드 데이터 수집 시 전체 코드가 아닌 높은 복잡도를 가진 코드 모듈 영역을 대상으로만 나쁜 코드 데이터를 수집한다. 이후 수집한 데이터셋을 CodeBERT 모델의 전이 학습하여 코드 공통 취약점을 탐색하는 방법을 제안한다. 해당 데이터셋을 통해 CodeBERT 모델이 코드의 공통 취약점 패턴을 더 잘 학습할 수 있다. 이를 통해 전통적인 방법보다 인공지능 모델을 이용해 코드를 분석하고 공통 취약점 패턴을 더 정확하게 식별할 수 있을 것으로 기대한다.

키워드 : 소프트웨어 공학, 코드 가시화, 코드 복잡도, 코드 취약점, 인공지능

1. 서 론

소프트웨어 가시화는 품질 지표에 대한 추출을 통해 저품질 코드 영역을 식별하고 이를 리팩토링 할 수 있도록 나타내어, 소프트웨어를 고품질화하는 기법이다. 현재 소프트웨어 가시화 기법은 대부분 사람에 의해 구현된 규칙을 통해 품질 지표를 추출한다. 따라서 새로운 품질 지표를 측정하기 위해 반드시 새로운 규칙을 구현해줘야만 한다. 새로운 규칙의 구현은 다양한 변수와 이에 따라 발생하는 다양한 경우의 수를 모두 고려해야 하므로 많은 시간과 노력이 필요한 작업이다.

현재 소프트웨어 공학과 인공지능을 접목하는 연구가 활발히 진행되고 있다[1]. 그 중, 소프트웨어 가시화 기법에 인공지능을 접목하여 고도화하는 연구가 진행되고 있다. 해당 연

※ 본 연구는 2023년도 문화체육관광부의 재원으로 한국콘텐츠진흥원(과제명: 인공지능 기반 사용자 대화형 멀티모달 인터랙티브 스토리텔링 3D장면 저작 기술 개발, 과제번호: RS-2023-00227917,기여율:50%) 지원과 2023년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업(과제명: NLP BERT Model 기반 자동 리팩토링을 통한 무결점 코드화 연구, 과제번호: No.2021R1I1A3050407,기여율:40%)의 지원과 2022년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업(과제명: 비정형 요구사항 명세서 기반 자동 비용 예측 및 역공학을 통한 검증 연구, 과제번호: No.2021R1I1A1A01044060,기여율:10%)의 지원을 받아 수행된 연구임.
※ 이 논문은 2023년 소프트웨어공학연구회의 우수논문으로 "Bad Code 패턴의 지도 학습을 통한 Bad Code 식별 적용 사례"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 홍익대학교 소프트웨어융합학과 석사과정
†† 정 회 원 : 홍익대학교 소프트웨어융합학과 초빙교수
††† 정 회 원 : 홍익대학교 소프트웨어융합학과 정교수
Manuscript Received : May 4, 2023
First Revision : July 27, 2023
Accepted : August 22, 2023
* Corresponding Author : R. Young Chul Kim(bob@hongik.ac.kr)

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/ licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

구는 인공지능이 코드의 패턴을 학습하고 소프트웨어를 가시화한다. 이후, 전통적인 소프트웨어 공학적인 방법의 소프트웨어 가시화와 인공지능을 통한 소프트웨어 가시화를 비교하고자 한다. 본 논문에서는 CodeBERT 모델에 나쁜 코드의 전이 학습을 통해 모델이 스스로 나쁜 코드 패턴을 학습하는 방법을 제안한다. 해당 방법을 코드에 대한 공통 취약점 목록(Common Weakness Enumeration)에 적용한다. 이를 통해 인공지능이 소스 코드로부터 공통 취약점의 패턴을 스스로 학습하고 식별할 수 있다.

논문의 구조는 다음과 같다. 2장에서는 기존 코드 가시화 연구, 기존 인공지능 기반 코드 취약점 학습 연구, CodeBERT 모델, 공통 취약점 목록 그리고 Programming Mistake Detector (PMD) 도구에 대해 설명한다. 3장에서는 본 논문의 주제인 CodeBERT 모델의 전이 학습 기반 코드 공통 취약점 탐색을 설명한다. 4장에서는 코드 공통 취약점 항목에 관한 적용 사례를 설명한다. 마지막으로 결론 및 향후 연구를 언급한다.

2. 관련 연구

2.1 코드 가시화 연구

코드 가시화는 코드를 분석하여 추출한 정보를 표와 다이어그램 등으로 가시화하는 것이다. 이를 통해 소프트웨어의 비가시적인 특성을 극복하여 소프트웨어 개발 프로젝트에 관련된 이해 관계자가 알기 쉽게, 소프트웨어의 각 설계 및 복잡도, 결함 부위에 대한 가이드 등을 나타낼 수 있다[2-6].

Fig. 1은 소프트웨어 가시화 연구에 인공지능을 접목하여 고도화하는 연구 과정과 내용을 요약하여 정리한 그래프이다. 해당 연구는 1차 연도에 소프트웨어의 품질 검증을 위한 정적 분석기 및 코드 가시화 기술을 개발하고, 2차 연도는 인공지능 모델에 대한 학습을 통한 무결점 소프트웨어 코드 자동 생성 기술을 개발하고, 3차 연도는 소프트웨어의 품질 역량 강화를 위한 무결점 코드 가시화 서비스를 구축한다.

Fig. 2는 기존 규칙 기반의 코드 가시화 툴 체인의 구조도이다[7]. 툴체인은 코드 분석 도구, 코드 복잡도 분석 도구 그리고 코드 가시화 도구 세 개의 도구로 이루어져 있다. 코드 분석 도구는 대상 프로그래밍 언어에 맞는 정적 분석 도구를 이용해 대상 코드를 분석 후 정보를 추출한다. 추출된 코드 정보는 데이터베이스에 저장된다. 코드 복잡도 분석 도구는 데이터베이스에 저장된 코드의 정보를 통해 객체지향 결합도와 응집도 등의 복잡도를 계산한다. 계산된 복잡도 또한 데이터베이스에 저장된다. 코드 가시화 도구는 데이터베이스에 저장된 코드의 정보를 바탕으로 설계와 복잡 모듈을 가시화한다.

코드 복잡도는 코드가 복잡한 정도를 나타내는 품질 지표이다. 코드 복잡도 분석 도구에서는 각 복잡도에 대한 규칙을 통해 복잡도를 계산한다. Fig. 3은 CK Metric[10] 중 하나인 Coupling Between Object(CBO)를 계산하는 규칙이다. CBO는 하나의 클래스가 연결된 다른 클래스의 개수이다. CBO를 계산하는 규칙은 클래스 간 상속, 구현, 연관 등의 관계를 찾아 클래스가 연결된 다른 클래스의 개수를 계산한다.

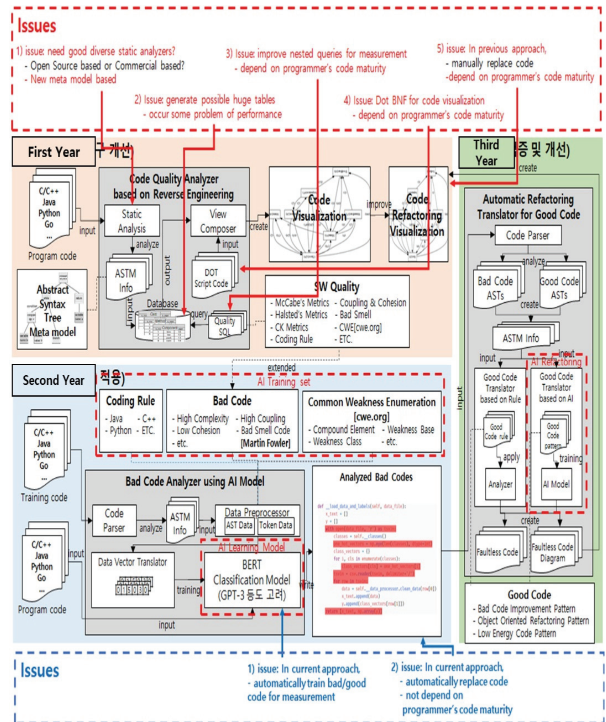


Fig. 1. Code Visualization Advancement Research

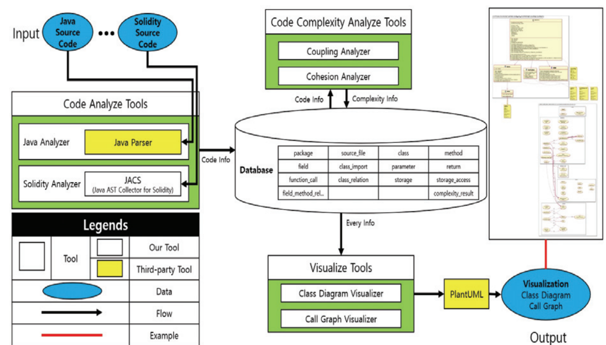


Fig. 2. Previous Code Visualization Tool Chain

```
private Vector<Integer> cbo() {
    Vector<Integer> cbo = new Vector<Integer>();
    try {
        Statement stat = con.createStatement();
        int numberOfClass = stat.executeQuery("SELECT COUNT(*) FROM class;").getInt(1);
        for(int i = 1; i < numberOfClass+1; i ++) {
            HashSet<Integer> relation = new HashSet<Integer>();
            ResultSet targetRS = stat.executeQuery("SELECT * "
                                + "FROM class_relationship "
                                + "WHERE origin_class_id = "+i+";");
            while(targetRS.next()) {
                relation.add(targetRS.getInt("target_class_id"));
            }
            targetRS.close();
            ResultSet originRS = stat.executeQuery("SELECT * "
                                + "FROM class_relationship "
                                + "WHERE target_class_id = "+i+";");
            while(originRS.next()) {
                relation.add(originRS.getInt("origin_class_id"));
            }
            originRS.close();
            cbo.add(relation.size());
        }
        stat.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return cbo;
}
```

Fig. 3. Example Code Complexity Rule

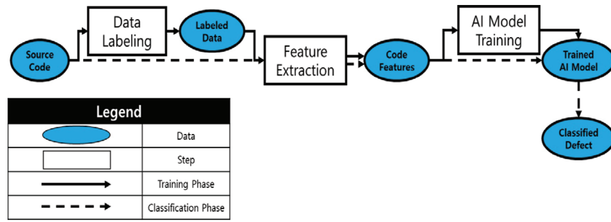


Fig. 4. Common Pipeline used for Defect Detection and Classification through AI Model

2.2 기존 인공지능 기반 코드 취약점 학습 연구

Fig. 4는 머신러닝을 통해 결함을 예측하는 모델의 일반적인 파이프라인이다[8]. 인공지능 모델에 결함을 학습하는 일반적인 과정은 다음과 같다. 우선 소스 코드를 클래스, 모듈, 함수 등의 단위로 쪼갠다. 이후 나누어진 소스 코드 단위의 특징을 추출하고, 취약점의 유무를 라벨링 한다. 이후 추출된 특징과 취약점으로 인공지능 모델을 훈련한다. 대부분의 연구는 Halstead Metric[9], CK Metric, McCabe's Cyclomatic Complexity[11] 등의 코드 품질 지표들을 소스 코드의 특징으로써 추출한다. 이외에도 Code Bad Smell[12], Abstract Syntax Tree를 특징으로써 학습에 이용하는 경우가 있으며, 코드 그 자체를 모델에 입력하여 사용하는 연구는 매우 드물다.

현재 많이 사용되고 있는 규칙 기반 결함 탐지 도구는 코드의 어떤 줄이 어떤 취약점에 노출되어 있는지 표시한다. 반면에 코드를 함수, 모듈, 클래스 단위로 쪼개어 특징을 추출하고 학습하는 방법은 코드의 어떤 부분이 결함의 원인인지 정확히 알기가 어렵다는 문제점이 있다.

2.3 CodeBERT 모델

BERT 모델은 Google에서 발표한 자연어 처리 모델이다[13]. Attention 메커니즘은 입력과 출력 간의 의존성을 모델링하고, 이를 통해 출력을 예측하기에 적합한 입력을 추천하는 메커니즘이다. Attention 층은 주로 기존의 RNN 및 LSTM 모델의 단기 기억 능력을 긴 시간 동안 유지하기 위한 용도로 사용되었다. Google에서는 Attention 층으로만 구성된 학습 모델인 Transformer 모델을 발표하였고, 이는 기존 Attention 층이 포함된 RNN 계열의 모델들보다 특히 자연어 처리 문제에서 개선된 성능을 보였다[14]. BERT는 Transformer 모델을 대용량의 데이터를 통해 사전 학습한 모델이며, 상대적으로 적은 양의 데이터를 통한 전이 학습으로도 높은 성능을 끌어낼 수 있다.

CodeBERT는 Transformer 모델에 대해 자연어가 아닌 6가지의 프로그래밍 언어 (Python, Java, JavaScript, PHP, Ruby, Go)에 대해 사전 학습한 모델이다[15]. 따라서 해당 모델에 대한 전이 학습을 통해 프로그래밍 언어와 관련된 문제를 해결하는 모델을 비교적 쉽게 학습할 수 있다.

2.4 공통 취약점 목록(Common Weakness Enumeration)

공통 취약점 목록은 커뮤니티를 통해 개발된 소프트웨어 및 하드웨어에서 일반적으로 식별할 수 있는 취약점들을 목록화 한 것이다[16]. 공통 취약점 목록은 프로그램에 대한 취약점의 식별 기준 및 취약점의 완화와 방지의 기준선의 역할을 한다. 소프트웨어 공통 취약점 목록은 소프트웨어의 설계, 아키텍처, 코드 등에 존재하는 취약점을 담고 있다.

2.5 Programming Mistake Detector

PMD는 프로그램의 취약점을 찾는 소스 코드 분석 오픈소스 도구이다[17]. Java, JavaScript 등 다양한 프로그래밍 언어에 대한 취약점 식별을 지원하며, Copy-Paste-Detector 도구가 내장되어 있어 중복 코드에 대해서도 탐지할 수 있다. PMD는 기본적으로 각 언어에 대해 다양한 취약점 규칙 들을 포함한다. 또한, 필요시 사용자 정의 규칙을 정의하여 새로운 취약점 혹은 사용자가 원하는 패턴의 코드를 식별할 수 있다.

3. CodeBERT 모델의 전이 학습 기반 코드 공통 취약점 탐색

Fig. 5는 CodeBERT 모델에 코드 공통 취약점을 전이 학습하는 과정의 구조도이다. CodeBERT 모델에 대한 전이 학습 과정에서는 크게 데이터셋 수집 단계, 데이터셋 전처리 단계 그리고 전이 학습 진행 단계가 있다.

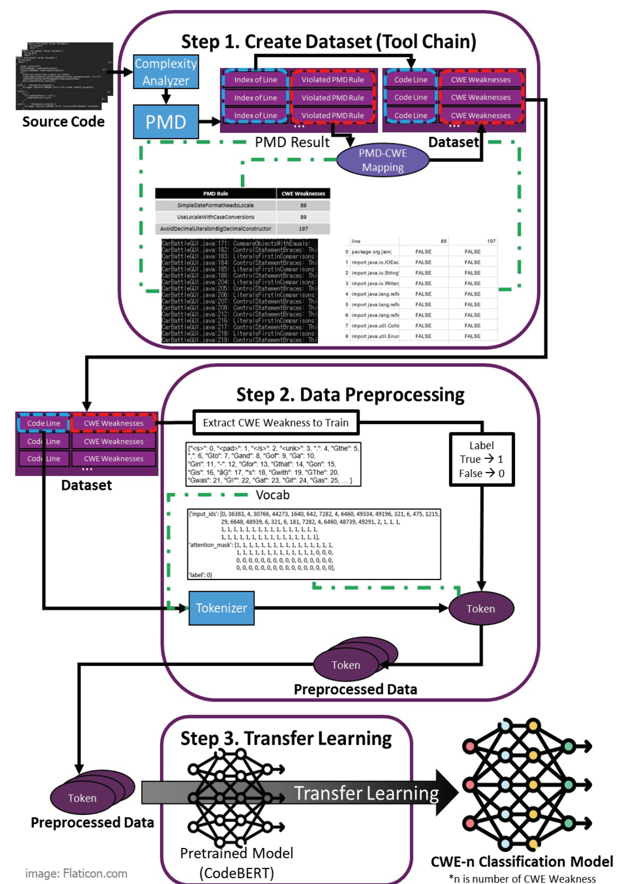


Fig. 5. Pipeline for Transfer Learning Bad Code Patterns to CodeBERT Model.

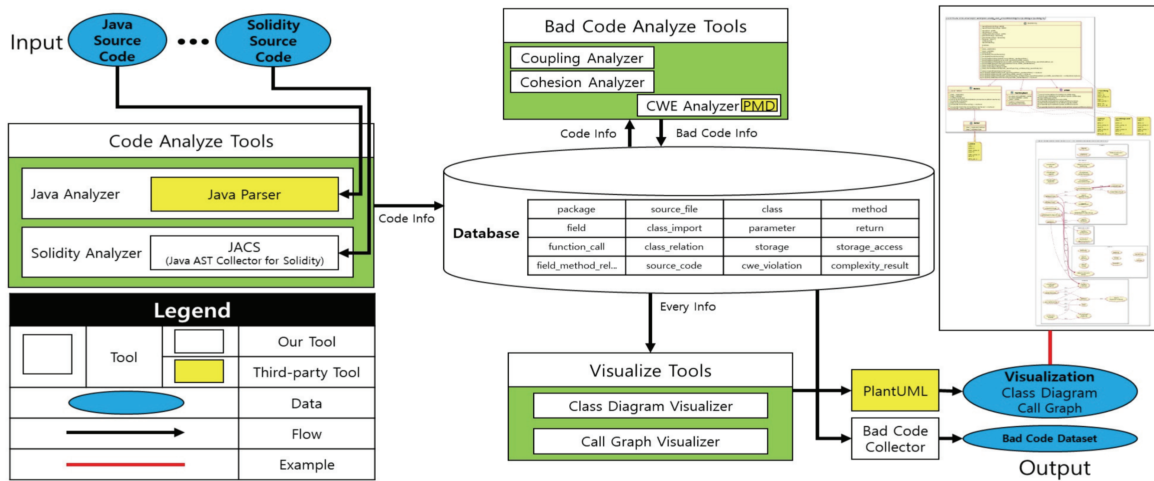


Fig. 6. Code Visualization Tool Chain with Bad Code Collector

3.1 데이터셋 수집

기존 나쁜 코드 패턴의 지도 학습을 통한 나쁜 코드 식별 연구[18]는 PMD 도구를 이용하여 데이터셋을 생성했다. 이 과정에서 PMD 도구 오탐지 한 코드 공통 취약점이 데이터셋에 포함될 수 있다. 오탐지 취약점은 인공지능 모델이 취약점의 패턴을 학습하는 것을 방해하고, 모델의 성능을 저해한다.

Fig. 6은 공통 취약점 검출 도구(CWE Analyzer)가 추가된 코드 가시화 툴체인이다[19]. 공통 취약점 검출 도구는 나쁜 코드 분석 도구에 포함되는 도구로서 PMD를 통해 코드로부터 취약점을 식별한다. 식별된 PMD 취약점은 PMD 취약점 목록과 공통 취약점 목록 간의 매핑 테이블을 통해 공통 취약점의 ID로 변환된다. 이후 공통 취약점이 발견된 코드 라인의 ID와 식별된 공통 취약점을 데이터베이스에 기록한다.

코드 복잡도는 코드의 취약점과 큰 상관관계 갖고 있다[20]. 또한 기존 인공지능 기반의 코드 취약점 학습 연구에서도 측정한 코드의 복잡도를 코드의 특징으로써 추출하여 학습 및 결함 예측에 사용했다. 따라서 수집하는 데이터셋의 오탐지 취약점 반영을 줄이고 신뢰도를 높이기 위해 코드 복잡도를 적용한다. 툴체인은 산출물로서 복잡도가 높은 클래스의 취약점 식별 결과를 데이터셋으로 만든다. 이를 통해 PMD 도구만을 사용하여 수집된 데이터셋보다 신뢰도가 높은 데이터셋을 수집할 수 있다.

3.2 데이터셋 전처리

CodeBERT 모델에 코드 라인의 공통 취약점 항목 패턴을 학습하기 위해 데이터셋을 전처리한다. 데이터 전처리 단계는 공통 취약점 항목 추출, 코드 라인 데이터 정제, 토큰화 그리고 클래스 균형 4개의 과정으로 나눌 수 있다.

공통 취약점 항목 추출 과정에서는 데이터셋에서 모델 학습에 필요한 데이터만 추출한다. CodeBERT 모델이 특정 공통 취약점 항목을 학습하는 것이 목적이기 때문에 학습할 공통 취약점 항목의 라벨과 코드 라인을 추출한다.

코드 라인 데이터 정제 과정에서는 취약점 식별과 관련 없는 노이즈를 제거하는 작업을 진행한다. 주석 혹은 공백만이 포함된 코드 라인은 공통 취약점과는 크게 연관이 없다. 따라서 이러한 노이즈를 학습 데이터셋으로부터 제거한다.

토큰화 과정에서는 CodeBERT 모델의 사전학습 Tokenizer를 통해 코드 라인을 토큰 시퀀스로 변환한다. 변환된 토큰 시퀀스에 대해 공통 취약점 라벨을 추가한다.

학습 데이터의 클래스 불균형 문제는 인공지능 모델의 학습을 저하하는 요인 중 하나이다. 수집한 데이터셋에는 취약한 코드 라인보다 취약하지 않은 코드 라인이 훨씬 많다. 이를 해결하기 위해 취약한 코드 라인 데이터를 증폭하는 오버샘플링 기법과 취약하지 않은 코드 라인의 개수를 줄이는 언더샘플링 기법을 적용할 수 있다.

3.3 CodeBERT 모델에 대한 전이 학습

앞선 두 단계를 통해 수집 및 전처리 된 학습 데이터셋을 통해 CodeBERT 모델에 대해 전이 학습을 진행한다. 이를 통해 CodeBERT 모델이 특정 공통 취약점 항목의 패턴 학습한다. 학습이 끝난 모델은 입력된 코드 라인에 대해 취약한 코드 라인인지 식별한다. 이때 모델이 과적합 되지 않고 잘 학습될 수 있도록 학습률, 학습 횟수 등을 조절할 수 있다.

4. 적용 사례

적용 사례로서 ‘CWE-400: Uncontrolled Resource Consumption’ 항목의 패턴을 CodeBERT 모델에 학습한다. PMD 만을 이용해 수집한 데이터와 툴체인을 통해 수집한 데이터로 각각 모델을 학습하고, Juliet Java 1.3[21]로 테스트한 결과를 비교한다.

Table 1은 PMD만을 이용해 수집한 데이터의 정보이다. PMD 도구를 이용해 9,280개의 깃허브 오픈소스 JAVA 프로젝트로부터 50,923개의 소스 코드에 대해 취약점의 검출 여부를 라벨링 했다. 이를 통해 총 11,893,126줄의 코드 라인에 대해 250,918줄의 취약점이 있는 코드 라인과 11,642,208줄의 취약점이 검출되지 않은 코드 라인으로 분류했다. True와 False로 라벨링 된 데이터 간 데이터 불균형을 언더샘플링 기법을 통해 균형을 맞췄다.

Table 1. Data Labeled by PMD

CWE Entry ID	Number of Labeled Data (Line)		Number of Total Data (Line)
400	True	250,918	11,893,126
	False	11,642,208	

Table 2. Data Labeled by Code Visualization Tool Chain

CWE Entry ID	Number of Labeled Date (Line)		Number of Total Date (Line)
400	True	7,981	10,705,865
	False	10,697,884	

Table 3. Confusion Matrix of PMD Dataset

		Classification Result	
		False	*True*
Juliet Java 1.3 Label	*Good Case*	1,596	4,440
	Bad Case	482	1,953

Table 4. Confusion Matrix of Code Visualization Tool Chain Dataset

		Classification Result	
		False	*True*
Juliet Java 1.3 Label	*Good Case*	719	5,317
	Bad Case	413	2,022

Table 2는 코드 가시화 툴체인을 통해 수집한 데이터의 정보이다. 툴체인을 통해 1,630개의 깃허브 오픈소스 JAVA 프로젝트로부터 66,504개의 클래스에 대해 취약점의 검출 여부를 라벨링 했다. 이를 통해 총 10,705,865줄의 코드 라인에 대해 7,981줄의 취약점이 있는 코드 라인과 10,705,865줄의 취약점이 검출되지 않은 코드 라인으로 분류했다. PMD를 통해 수집된 데이터와 마찬가지로 True와 False로 라벨링 된 데이터 간 불균형이 매우 심하므로 언더샘플링 기법을 통해 균형을 맞췄다.

학습된 모델들의 테스트 방법으로서 Juliet Java 1.3을 이용한다. 해당 데이터셋은 National Institute of Standards and Technology에서 배포하는 공통 취약점 목록에 대한 테스트 케이스 묶음으로서 보안 도구 및 취약점 검출 도구에 대한 검증 데이터셋으로써 보편적으로 사용된다. Juliet Java 1.3에는 각 공통 취약점 항목에 대해 나쁜 경우와 좋은 경우의 함수로 나뉜다. 이때 나쁜 경우의 함수에는 나쁜 Source로부터 입력된 데이터를 나쁜 Sink를 통해 처리하는 경우가 포함된다. 좋은 경우의 함수에는 나쁜 Source로부터 입력된 데이터를 좋은 Sink로 처리하는 경우 혹은 좋은 Source로부터 입력된 데이터를 나쁜 Sink를 통해 처리하는 경우가 포함된다. 따라서 좋은 경우의 함수임에도 불구하고 취약점 요소가 포함된 경우가 많다. 따라서 모델은 좋은 경우의 함수에 포함된 취약점 라인을 검출하고, 나쁜 경우의 함수로 구분할 수 있다. 따라서 모델 테스트 시 나쁜 함수에 대한 취약점 식별 정확도에 대해서만 고려한다.

Table 3은 PMD 도구를 통해 수집한 데이터로 학습한 모델의 검증 혼동행렬이다. Table 4는 코드 가시화 툴체인을 통해 수집한 데이터로 학습한 모델의 검증 혼동행렬이다. 식별 정확도는 모든 Bad Case에 대해 True로 분류한 Bad Case의 비율이다. Table 3 혼동행렬에서의 식별 정확도는 약 80.21%이다. Table 4 혼동행렬에서의 식별 정확도는 약 83.04%이다. 두 모델 모두 언더샘플링을 통해 데이터 불균형 문제를 해결했다. 학습에 이용된 PMD 데이터셋은 501,836줄이며, 코드 가시화 툴체인 데이터셋은 15,962줄이다. 툴체인으로 수집된 데이터셋으로 학습한 모델이 더 적은 양의 데이터로 학습했음에도 불구하고, 나쁜 케이스의 함수에 대해 더 취약점을 잘 검출했다.

5. 결 론

본 논문에서는 CodeBERT 모델에 나쁜 코드의 패턴을 전이 학습하여 나쁜 코드를 식별하는 방법을 제안하였다. 이를 공통 취약점에 적용하여 취약점의 패턴을 모델에 학습했다. 학습한 모델을 Juliet Java 1.3에 포함된 함수를 분류하여 식별 정확도를 테스트하였다. 이러한 방법을 통해 CodeBERT 모델이 취약점의 패턴을 학습할 수 있다. 또한 복잡도가 높은 클래스에 대해 수집한 데이터를 통해 인공지능 모델이 취약점의 패턴을 더 잘 학습할 수 있다. 이를 통해 새로운 유형의 취약점 혹은 규칙을 정하기 까다로운 취약점에 대해 CodeBERT 모델이 패턴을 스스로 학습하는 것을 기대한다. 추후 더 많은 데이터에 대한 학습을 통해 모델이 식별할 수 있는 나쁜 코드의 종류를 늘이고 정확도를 향상할 것이다. 이때 데이터셋의 정확도 향상을 위해 복잡도 외의 추가적인 지표를 기준으로 나쁜 코드를 수집하는 것을 고려한다. 이후 이를 규칙 기반의 도구와 비교를 통해 성능을 비교 및 검증하는 연구를 할 것이다. 또한, 나쁜 코드가 식별된 모듈 영역에 대해 인공지능으로 리팩토링하는 연구도 진행할 예정이다.

References

[1] A. D. Carleton, E. Harper, T. Menzies, T. Xie, S. Eldh, and M. R. Lyu, "The AI effect: Working at the Intersection of AI and SE," *IEEE Software*, Vol.37, No.4, pp.26-35, 2020.

[2] G. H. Kang, R. Y. C. Kim, G. S. Yi, Y. S. Kim, Y. B. Park, and H. S. Son, "A study on code static analysis with open source-based tool chainization," *KIISE Transactions on Computing Practices*, Vol.21, No.2, pp.148-153, 2015.

[3] H. Kwon and R. Y. C. Kim, "Extracting use case design mechanisms via programming based on reverse engineering," *International Journal of Applied Engineering Research*, Vol.10, No.90, pp.503-505, 2015.

[4] B. K. Park, G. H. Kang, H. S. Son, B. K. Jeon, and R. Y. C. Kim, "Code visualization for performance improvement of Java code for controlling smart traffic system in the smart city," *Applied Sciences*, Vol.10, No.8, 2020.

[5] S. J. Jung, J. H. Kim, W. Y. Lee, B. K. Park, H. S. Son, and R. Y. C. Kim, "Automatic UML design extraction with software visualization based on reverse engineering," *International Journal of Advanced Smart Convergence*, Vol.10, No.3, pp.89-96, 2021.

[6] W. Y. Lee and R. Y. C. Kim, "Best practices on validation and extraction of object oriented designs with code visualization tool-chain," *Journal of Internet Computing and Services*, Vol.23, No.2, pp.79-86, 2022.

[7] C. S. Park, S. Y. Moon, and R. Y. C. Kim, "Quality visualization of quality metric indicators based on table normalization of static code building information," *KIPS Transactions on Software and Data Engineering*, Vol.12, No.5, pp.199-206, 2023.

[8] T. Sharma, M. Kechagia, S. Georgiou, R. Tiwari, I. Vats, H. Moazen, and F. Sarro, "A survey on machine learning techniques for source code analysis," *arXiv preprint arXiv:2110.09610*, 2021.

[9] Halstead, Maurice H., "Elements of software science (Operating and programming systems series)," Elsevier Science Inc., 1977.

[10] S. R. Chidamber, C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, Vol.20, No.6, pp.476-493, 1994.

[11] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, Vol.4, pp.308-320, 1976.

[12] M. Fowler, "Refactoring," Addison-Wesley Professional, 2018.

[13] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[14] A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems*, 30, 2017.

[15] Z. Feng et al., "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[16] Common Weakness Enumeration [Internet], http://cwe.mitre.org

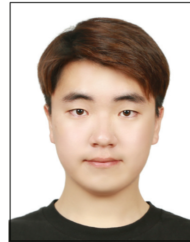
[17] PMD [Internet], https://pmd.github.io/

[18] C. S. Park, J. H. Kim, S. Y. Moon, and R. Y. C. Kim, "Applied practice on identifying bad codes through supervised learning with bad code patterns," *Proceedings of the 25th Korea Conference on Software Engineering*, Vol.25, No.1, pp.119-120, 2023.

[19] C. S. Park, W. S. Jang, and R. Y. C. Kim, "Tool Chain Mechanism with Identifying and Collecting High Quality Data for Learning Bad Code based on Code Visualization," *2023 Conference of KISM*, Vol.12, No.1, pp.52-53, 2023.

[20] N. Medeiros, N. Ivaki, P. Costa, and M. Vieira, "Software Metrics as Indicators of Security Vulnerabilities," *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, Toulouse, France, pp.216-227, 2017.

[21] NSA Center for Assured Software, 2017, "Juliet Java 1.3," [Online].Available:https://samate.nist.gov/SARD/test-suites/111



박 찬 솔

https://orcid.org/0009-0009-9462-1783
e-mail : c2193102@g.hongik.ac.kr
2022년 홍익대학교 소프트웨어융합학과 (학사)
2022년 ~ 현 재 홍익대학교 소프트웨어융합학과 석사과정
관심분야 : Software Visualization, Software Quality

문 소 영

https://orcid.org/0009-0000-9498-9689
e-mail : whit2@hongik.ac.kr
2007년 ~ 2012년 ㈜액트
2018년 홍익대학교 전자전산공학과(석・박사)
2017년 ~ 현 재 NIPA SP 선임심사원
2019년 ~ 현 재 홍익대학교 소프트웨어융합학과 초빙교수
관심분야 : Software Visualization, Requirement Engineering

김 영 철

https://orcid.org/0000-0002-2147-5713
e-mail : bob@hongik.ac.kr
2000년 IIT, Dept. of Computer Science (박사)
2000년 ~ 2001년 LG산전 중앙연구소 Embedded System 부장
2001년 ~ 현 재 홍익대학교 소프트웨어융합학과 정교수
관심분야 : Software Visualization, TMM, Requirement