ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 49, No. 8, pp. 633-645, 2022. 8
https://doi.org/10.5626/JOK.2022.49.8.633

오픈 소스 기계학습 애플리케이션에 대한 결함 사례 조사

(An Empirical Study on Defects in Open Source Artificial Intelligence Applications)

최 윤 호 † (Yoon Ho Choi)　이 창 공 †† (Changgong Lee)　남 재 창 ††† (Jaechang Nam)

요 약 기계학습 기반 프로그래밍 패러다임과 전통적인 방식의 프로그래밍 패러다임의 차이는 기계학습 애플리케이션에서 발생할 수 있는 결함을 검출하고 이해, 분석, 해결하는 것에 다른 양상을 나타낼 수 있다. 이와 같은 상황에서, 본 연구는 기계학습 기반 시스템이 가진 결함을 이해하고 분석하기 위해, 오픈 소스 기계학습 애플리케이션에서 발생했던 결함의 사례들을 수집하고 빈번하게 발생하는 결함의 원인을 파악하고자 하였다. 이를 위해, GitHub에 공개된 10개의 오픈소스 기계학습 애플리케이션을 대상으로 GitHub 이슈 게시판에 있는 1,205개의 결함 이슈 보고와 결함 수정 코드 이력을 직접 분석하여 보고/발견/수정되었던 결함에 대해 분석하였다. 10개 중 5개 이상의 프로젝트에서 공통으로 발견된 결함의 근본적인 원인 기준으로 20개의 결함 원인 범주를 설정하였다. 본 연구의 결과는 결함 위치 추적, 가능한 결함 해결 코드 수정 제안 등의 품질 향상 기술에 활용될 수 있을 것으로 기대된다.

키워드: 기계학습, 소프트웨어 결함, 실증적 조사, 소프트웨어 저장소 마이닝

Abstract The differences between the programming paradigm of applications using artificial intelligence (AI) and traditional applications may show different results in detecting, understanding, analyzing, and fixing defects. In this study, we collect defects that have been reported in open source AI applications and identify common causes of the defects to understand and analyze them in AI-based systems. To this end, we analyze the defects of ten open-source AI applications archived on GitHub by inspecting 1,205 issues and defect-fixing code changes that had been reported, found, and fixed. We classified the defects into 20 categories based on their causes, which are found in at least five out of ten projects. We expect that the result of this study will provide useful information in software quality assurance approaches such as fault localization and patch suggestion.

Keywords: artificial intelligence, software defect, empirical study, mining software repositories

· 본 연구는 한국과학기술정보연구원(KISTI)의 위탁연구 과제(P22031)와 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(No.2021R1F1A1063049)으로 수행된 연구임
· 이 논문은 2022 한국 소프트웨어공학 학술대회에서 '오픈 소스 기계학습 어플리케이션에 대한 결함 사례 조사'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : 한동대학교 정보통신공학과 학생
yhchoi@handong.ac.kr
†† 비 회 원 : 한동대학교 전산전자공학부 학생
zackcglee@handong.ac.kr
††† 정 회 원 : 한동대학교 전산전자공학부 교수(Handong Global Univ.)
jcnam@handong.edu
(Corresponding author임)

논문접수 : 2022년 3월 25일
(Received 25 March 2022)
논문수정 : 2022년 5월 17일
(Revised 17 May 2022)
심사완료 : 2022년 5월 19일
(Accepted 19 May 2022)

정보과학회논문지 제49권 제8호(2022. 8)

1. 서 론

기계학습 기법이 등장한 이후, 음성 인식, 이미지 처리, 자연어 처리, 자율 주행 등 여러 가지 문제에 대해 전통적으로 사용했던 알고리즘보다 뛰어난 성능을 보이며 여러 분야에 적용되고 있다[1-4]. 특히, 컴퓨터의 연산 능력 증대와 접근할 수 있는 데이터의 방대한 증가를 통해 기계학습 기법 중 하나인 딥러닝 기술이 폭넓은 분야에서 높은 성능을 보이며 사용되고 있다. 기계학습 기법을 사용한 소프트웨어 시스템의 증가로 인해, 기계학습 시스템 분석 및 개선의 필요성이 더욱 중요한 문제로 나타나고 있다.

기계학습 기법을 기반으로 한 프로그래밍 패러다임은 전통적인 방식의 프로그래밍 패러다임과 큰 차이가 있다. 전통적인 방식의 프로그래밍은 풀고자 하는 문제에 적합하게 작동할 수 있는 프로그램 모델을 작성하여 문제를 직접적으로 해결하는 방식의 프로그래밍을 하였다. 하지만 기계학습 애플리케이션의 경우, 문제를 직접적으로 해결하는 프로그램을 작성하는 방식보다도, 문제를 해결하기 위한 기계학습 모델의 신경망 구조를 프로그래밍하는 방식으로 프로그램을 작성한다. 또한, 문제를 해결하기 위해 작성된 모델은, 직접적인 문제 해결 알고리즘을 통해 동작하는 것이 아니라, 방대한 양의 데이터를 학습하여 문제를 해결한다. 이러한 특성을 기반으로, 기계학습 모델을 작성함에 있어 모델의 신경망 구조와 모델의 훈련 과정은 프로그램을 작성하는 개발자의 선택에 따라 설정해야 한다. 이와 같이, 기계학습 모델을 작성하는 복잡한 신경망 구조 설정과 노드의 계층 구성 등과 같은 문제는 전통적인 소프트웨어 개발 과정에서 겪게 되는 문제와는 다른 성격을 보이게 된다[5]. 또한, 훈련 과정은 러닝 레이트(learning rate)와 드롭아웃 레이트(dropout rate)와 같은 하이퍼 파라미터를 고려해 설정을 바꿔가며 계속 반복하는 과정을 거치게 된다.

기계학습 기반 프로그래밍 패러다임과 전통적인 방식의 프로그래밍 패러다임의 차이는 기계학습 애플리케이션에서 발생 가능한 결함을 기존의 방식으로 검출하고 이해, 분석, 해결하는 것에 다른 양상을 보일 수 있다. 기계학습 애플리케이션이 적용되고 있는 범위가 일반적인 소프트웨어 시스템부터 미션 크리티컬 시스템까지 광범위한 상황 가운데, 특히나, 미션 크리티컬 시스템에서 사용된 기계학습 모델의 결함은 막대한 피해를 끼칠 수 있다. 예를 들면, 자율 주행 시스템에서 물체 인식 모델의 결함은 인명 피해로 이어질 수 있으며, 얼굴인식 모델을 적용한 보안 시스템은 경제적 피해뿐 아니라 개인 정보 유출 등의 피해로 이어질 수 있다. 이를 방지하기 위해, 기존의 프로그래밍 패러다임과 다른, 기계학습 모델이 가진 결함에 대한 분석과 이해가 요구된다.

소프트웨어 시스템에서 발생하는 결함을 이해하고 분석하는 효과적인 방식 중 하나는 기존에 발생했던 결함에 대해 분석하고 이해하는 것이며, 다양한 연구에 이와 같은 방식을 통해 결함에 대해 분석하였다[6-8]. 결함에 대해 보다 자세하고 면밀한 분석을 위해, 개발 과정에서 보고 및 발견된 다양한 결함의 여러 특성을 이해해야 한다. 또한, 이러한 분석과 이해를 바탕으로 결함 검출 도구와 결함 보고 분류, 결함 위치 추적, 가능한 결함 해결 코드 수정 제안, 테스팅/디버깅 비용 측정, 개발 과정 관리 등에 유용하게 사용할 수 있는 정보를 얻을 수 있다.

본 연구에서는 기계학습 기반 시스템이 가진 결함을 이해하고 분석하기 위해, 오픈소스 기계학습 애플리케이션에서 발생했던 결함의 사례들을 수집하고 기계학습 기반 시스템에서 빈번하게 발생하는 결함의 원인을 파악하고자 하였다. 이를 위해, GitHub에 공개된 10개의 오픈소스 기계학습 애플리케이션을 대상으로 GitHub 이슈 게시판에 있는 내용과 결함 해결 코드 수정 이력을 직접 분석하여 보고/발견/수정되었던 결함에 대해 분석하였다.

조사 대상으로 선정한 10개의 애플리케이션에서 발견한 1,205개의 결함 이슈 보고와 결함 수정 코드를 직접 분석해 10개 중 5개 이상의 프로젝트에서 공통적으로 발견된 결함의 근본적인 원인 기준으로 범주화하였으며, 20개의 결함 원인 범주를 설정하였다. 조사 결과, 가장 빈번하게 발생한 결함의 원인은 '예외적인 데이터 처리에 대한 누락'이었으며 214개의 결함 보고 내역이 존재하였다. 또한, '기본 값 및 옵션값의 잘못된 설정 및 누락', '라이브러리 임포트 및 설치 문제', '오타'를 원인으로 한 결함이 10개 중 9개의 애플리케이션에서 보고되었다. 본 연구의 결과를 바탕으로 기계학습 애플리케이션의 결함을 자동으로 검출하는 결함 검출 도구를 구현할 수 있으며, 연구 결과를 기계학습을 통한 소프트웨어 분석 및 결함 검출 기법의 훈련 데이터로 사용할 수 있다.

2. 관련 연구

머신러닝과 딥러닝 기술을 사용하는 소프트웨어에서 발생할 수 있는 결함에 대해 이해하고 분석하기 위해 진행되었던 이전 연구가 존재한다. 표 1은 관련 연구의 조사 대상, 결함 분석 방법, 분석한 결함 사례의 개수를 나타낸다.

Thung et al.은 머신러닝을 사용하는 세 가지의 시스템에 대해 결함/이슈 추적 시스템인 지라를 통해 보고된 결함을 바탕으로 직접 분석하였다. 연구에 사용된 시스템은 Mahout, Lucene, OpenNLP로 각 시스템에서 보고된 결함 이슈 총 1,950개 중 무작위로 500개를 선

표 1 관련 연구의 결함 사례 분석 대상과 방법, 분석한 결함 사례 개수
Table 1 Studied target, approach, and the number of analyzed defects of related works

	Thung et al. [9]	Zhang et al. [10]	Zhang et al. [5]	Islam et al. [11]
Target	Three AI-based OSS	4,960 of failed learning jobs on Philly	One deep learning library (Tensor Flow)	Five deep learning libraries
Approach	Analyzing GitHub issue	Analyzing failure cases of deep learning models by using Philly	Analyzing GitHub commit history and Stack Overflow	Analyzing GitHub commit history and Stack Overflow
# of analyzed defects	500	400	175	970

택하여 분석하였다. 분석 결과 총 11개의 범주로 결함을 구분하였다[9]. 본 연구에서는, GitHub 이슈를 통해 10개의 애플리케이션에서 보고된 1,205개의 결함 이슈를 분석함으로 관련 연구보다 다양한 분야에서 적용되고 있는 애플리케이션과 다양한 결함 이슈를 분석하였다는 점에서 차이가 있다.

Zhang et al.은 Philly 원격 딥러닝 플랫폼을 통해 딥러닝 모델 실행 결과 “FAILED” 상태로 실행이 종료되는 경우를 바탕으로, 실패하는 실행에 대한 결함의 원인과 빈도를 파악하였다[10]. 총 4,960개의 실패한 실행 데이터를 수집하였으며, 그 중 400개의 결함 사례에 대해 분석해 20개의 결함 원인을 분류하였다. 본 연구에서는 GitHub 결함 보고를 기반으로, 런타임에서 실패하는 실행뿐 아니라, 성능 저하, 의도하지 않은 결과, 빌드 실패 등 보다 다양한 결함 증상에 대한 원인을 분석하였다는 점에서 차이가 있다.

Zhang et al.는 파이썬 딥러닝 라이브러리인 텐서플로를 사용한 딥러닝 기반 애플리케이션에 대해 스택 오버플로와 GitHub 커밋 이력을 바탕으로 발생했던 결함에 대해 조사하였다. 스택 오버플로를 통해 87개의 결함 데이터를 수집하였으며, GitHub 커밋 이력을 바탕으로 88개의 결함 데이터를 수집하여, 총 175개의 결함에 대해 분석하였다. 분석 결과, 총 7개의 결함 원인 범주와 4개의 결함 증상 범주로 결함을 분류하였다[5]. 본 연구에서는 텐서플로 뿐만 아니라, 다양한 파이썬 기반 딥러닝 라이브러리를 사용하는 애플리케이션에 대해서 조사하였으며, 자바, C++ 등의 프로그래밍 언어로 작성된 기계학습 모델에 대해 조사했다는 측면에서 더욱 다양한 환경에서 발생할 수 있는 결함에 대해 조사하였다.

Islam et al.은 스택 오버플로와 GitHub 커밋 이력을 활용해 딥러닝 라이브러리를 사용함에 있어 발생했던 결함에 대해, 결함의 종류와 원인, 증상으로 구분하여 각각 5개, 10개, 6개의 범주로 분류하였다. Caffe와 Keras, Tensorflow, Theano, Torch 총 5개의 딥러닝 라이브러리를 대상으로 하였으며 스택 오버플로의 게시글 2,716개를 바탕으로 찾은 415개의 결함 정보와 GitHub 커밋 이력 500개를 바탕으로 찾은 555개의 결함 정보를 분석하였다[11]. 본 연구에서는 GitHub 이슈를 이용해서 결함 정보를 수집하였으며 관련 연구보다 많은 숫자의 결함에 대해 분석하였다는 차이가 있다.

3. 연구 방법

3.1 개요

기계학습 코드에서 반복적으로 결함을 유발하는 원인에 대해 수집하고 분석하기 위해, 대표적인 소프트웨어 저장소인 GitHub에서 오픈소스 프로젝트로 개발된 기계학습 애플리케이션의 역사성 데이터를 활용해 결함 보고, 결험 해결 커밋 이력을 수집 및 분석함으로써, 결함 원인을 분석하고 범주화하였다.

그림 1은 본 연구의 조사 대상 선정 및 분류 과정을 간략하게 도식화하여 보여준다. 그림 1의 (1)과 같이, 조사 대상 선정을 위해 GitHub의 기계학습 프로젝트 중 닫힌 이슈의 개수가 500개 이상인 프로젝트를 선정한다. 조사 대상 프로젝트가 선정되면, 그림 1의 (2)와 같이 분석 대상이 될 수 있는 이슈를 분류하는 단계를 진행한다. GitHub에서 관리되고 있는 이슈는 결함과 무관한 내용도 포함하기 때문에, 결함과 직접적인 연관이 있는 이슈를 필터링한다. 그림 1의 (2)-1, (2)-2, (2)-3이 이슈를 필터링하는 각 단계를 도식화해 나타낸다. 첫 째로, 이슈의 라벨을 바탕으로 이슈를 필터링하고, 남은 이슈 중 이슈 내용을 통해 결함과 연관된 이슈만을 선정한다. 셋 째로, 남아 있는 이슈 중 해당 이슈를 수정한 코드 수정을 기반으로 결함 유발 원인을 파악하고 결함과 무관한 경우 필터링한다. 이와 같은 필터링 절차를 거친 이후 선정된 이슈에 대해서 결함 유발 원인을 기준으로 결함 사례를 분석하고 분류했다.

3.2 결함 유발 코드 사례 수집

3.2.1 조사 대상

3.2.1.1 조사 대상 선정 기준

사례 수집에 앞서 분석의 정밀도 향상을 위해 GitHub에 공개된 오픈 소스 프로젝트 중, 머신러닝 혹은 딥러닝 기술을 사용하였고 해당 저장소의 닫힌 이슈(Closed-

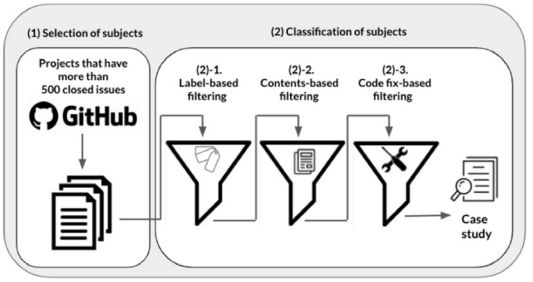

그림 1 본 연구의 조사 대상 분류
Fig. 1 Subject selection and issue classification process

Issue)의 개수가 500개 이상인 프로젝트를 조사 대상으로 선정하였다.

닫힌 이슈의 개수로 조사 대상의 기준을 정한 이유는 결함의 원인을 분석하기 위해 이슈를 통해 보고된 결함 보고가 필요하기 때문이다. 열려 있는 이슈(Opened-Issue)의 경우에는 해당 이슈가 결함과 명확히 관련되어 있는지 파악하기 어렵다는 문제가 있기 때문에 기준으로 선정하지 않았다. 또한, 닫힌 이슈 중에도 결함과 관련이 없는 이슈가 존재한다. 이 경우는 오픈 소스 프로젝트 사용에 대한 사용자들의 질문, 결함이 아님에도 불구하고 결함으로 오인하고 보고하는 경우, 소프트웨어 기능 향상에 대한 요구 등 개발자와 사용자의 상호 작용이 이슈에서 통합적으로 관리되고 있기 때문에 발생한다. 따라서, 닫힌 이슈의 개수가 일정한 정도 이상 있는 경우, 결함을 관리하는 이슈의 개수가 더욱 많이 존재할 것이라고 판단해 조사 대상 기준을 세웠다.

3.2.1.2 조사 대상 수집 방법 및 선정된 조사 대상

조사 대상 기준에 적합한 기계학습 오픈소스 프로젝트를 찾기 위해, GitHub에서 관리되고 있는 여러 오픈 소스 프로젝트 중 GitHub에서 제공하는 프로젝트 검색 태그에 Machine learning 또는 Deep learning 태그를 붙여놓은 프로젝트를 일차적으로 선별하였다. 이후, 해당 프로젝트가 머신러닝 혹은 딥러닝 기술을 사용하는 프로젝트인지 프로젝트 설명 문서와 코드를 참고하여 2차 선별하였다. 또한, 닫힌 이슈의 개수가 500개 이상인지 확인하기 위해 2차 분류된 프로젝트에 GitHub API 버전 4를 NodeJS와 함께 사용해 닫힌 이슈 개수와 이슈 제목, 이슈 라벨, 이슈 내용, 이슈 답글, 커밋 아이디, 커밋 메시지를 마이닝하였다.

최종적으로 조사 대상으로 선정한 프로젝트는 표 2와 같다. 총 10개의 프로젝트를 수집하였으며, 표 2의 각 행은 프로젝트 이름, 프로젝트가 이용되고 사용되는 분야, 닫힌 이슈의 개수를 나타낸다. 다양한 분야에 응용되고 있는 기계학습 코드를 분석하면서 일반성을 높이기 위해 응용 분야가 상이한 프로젝트를 선정하고자 하였다. 선정된 10개의 프로젝트의 응용 분야는 음성(Voice) 인식, 신체 인식, 자연어 처리, 광학 문자 인식, 물체 인식, 실시간 스트리밍 방송 미디어 솔루션, 챗봇, 사진첩 관리 솔루션, 기계학습 및 데이터 과학 애플리케이션 분석 및 공유 라이브러리로 다양한 분야에서 사용되고 있는 프로젝트로 구성하였다.

3.2.2 결함 원인 수집 방법

GitHub 이슈에서 결함이 보고/해결된 이슈를 찾아 분석함으로써 빈번하게 발생하는 결함의 원인에 대해 범주화하기 위해 총 두 단계의 필터링 과정을 거쳤으며, 필터링 이후 단계에서 남아있는 결함 이슈에 대해 일반적인 원인을 기반으로 분류하였다. 분류 단계에서는 결함과 관련된 이슈 내용과 해당 결함을 해결한 코드 수정을 직접 확인하였다. 첫째로, 이슈 라벨을 활용해 결함과 관련이 있는 이슈와 그렇지 않은 이슈를 분류하고 결함과 무관한 라벨이 붙어 있는 이슈를 제거하였다. 두 번째 단계로, 이슈 내용을 활용해 개발자가 이슈로 제기

표 2 본 연구의 조사 대상으로 선정한 프로젝트 목록
Table 2 Studied open source projects

No.	Application	Domain	# of closed issues
1	DeepSpeech [12]	speech recognition	1,905
2	OpenPose [13]	video/image processing	1,559
3	spaCy [14]	natural language processing	4,598
4	Real-Time-Voice-Cloning [15]	audio processing	646
5	tesseract [16]	optical character recognition	1,817
6	yolov5 [17]	object detection	3,207
7	mediapipe [18]	just-in-time streaming broadcast media solution	2,159
8	ChatterBot [19]	chatbot	1,240
9	photoprism [20]	a gallery management solution	833
10	streamlit [21]	a library for analyzing and sharing machine learning and data science application	1,393

된 결함에 내용에 동의하고 수정하였는지 확인해, 결함과 관련된 이슈 라벨이 붙어 있는 이슈가 실제로 결함과 관련된 이슈인지 분류하고 실제 결함이 아닌 경우는 제거하였다. 세 번째 단계로, 결함을 유발한 원인이 무엇인지 파악하고 어떻게 수정하였는지에 따라 결함의 원인을 분류하고 범주화하기 위해 결함 유발 코드를 직접 확인하고 분류하였다.

3.2.2.1 1차 분류: 이슈 라벨 기반 필터링

일차적으로 이슈의 라벨을 기반으로 한 분류를 진행하였다. GitHub에서 이슈는 개발자-개발자/개발자-사용자/사용자-사용자가 서로 소통할 수 있는 창구의 역할을 한다. 사용자는 개발자에게 이슈를 통해, 프로젝트를 실제로 활용하는 기술적인 방법에 대해서 질문할 수도 있으며, 결함이 있는 경우 결함을 보고할 수도 있다. 이처럼 다양한 이유로 이슈를 제기하는 상황에서 프로젝트를 원활히 관리하기 위해, 프로젝트 개발자들은 각 이슈에 대해 라벨을 붙여 이슈들을 관리한다. 예를 들면, 결함에 관련된 것은 'bug/fault' 등의 라벨을 사용하며, 중복된 이슈에 대해서는 'duplicate' 등의 라벨을 사용한다. 사용자의 질문/기술적 도움에 관한 이슈인 경우, help wanted 등의 라벨을 사용할 수도 있다.

본 연구의 목적은 결함 유발 코드 수정 사례를 수집하는 것이므로, 닫힌 이슈 중 결함과 관련된 라벨이 붙어 있는 이슈들로 1차 분류하였다. 프로젝트 이슈 관리를 위해 사용되는 라벨은 프로젝트마다 상이하게 설정이 가능하기 때문에, 조사 대상으로 선정한 10개의 프로젝트의 이슈 라벨 중 결함과 관련된 키워드인, 'fault'나 'bug' 키워드의 포함 여부를 바탕으로 라벨을 결함과 관련 있는 것과 그렇지 않은 것으로 구분하였다. 확인한 결함 관련 라벨을 바탕으로, 결함과 관련이 있는 라벨이 붙어있지 않은 모든 이슈는 조사 대상에서 제외하였다.

3.2.2.2 2차 분류: 이슈 내용 기반 필터링

두 번째 분류로서 이슈의 내용을 기반으로 한 필터링을 진행하였다. 결함과 관련된 라벨이 붙어 있는 이슈 중 결함이 아닌 것에 대해 결함이라고 라벨을 붙이거나, 중복된 내용이 서로 다른 결함 이슈로 오인되어 관리되는 때도 있다. 이와 같은 문제가 발생하는 이유 중 한 가지 이유는 개발자가 아닌 사용자가 직접 라벨을 붙여서 이슈 보고를 할 수 있도록 허용된 프로젝트의 경우에, 사용자가 결함 의심 이슈를 보고했지만, 개발자가 해당 이슈를 결함 이슈로 동의하지 않았기 때문이다. 다른 이유로는 서로 다른 증상으로 발현된 하나의 결함이 서로 다른 결함으로 오인되어 중복된 이슈로 처리되지 않을 수 있기 때문이다. 단순히 실수로 라벨을 잘못 붙이는 경우도 있을 수 있다.

결함으로 라벨이 붙어있지만 실제로 결함과 무관한 이슈의 해결 사례를 수집하는 것은 본 연구의 목적인 결함 유발 코드 수정 사례를 수집하는 것에 부합하지 않기 때문에, 이슈의 내용과 개발자의 답변 내용을 직접 확인하고 결함 라벨이 잘못 부여된 이슈들을 2차 분류하고 실제 결함이 아닌 이슈는 조사 대상에서 제외하였다. 또한, 중복된 이슈는 필터링하여 대표 이슈 한 가지만을 고려하였다. 표 3은 조사 대상이 된 각 프로젝트의 필터링 단계별 이슈 개수를 나타낸다.

3.2.2.3 3차 분류: 결함을 해결한 코드 수정 기반 분류

마지막 분류는 이슈를 해결한 코드 수정에 기반하여 진행하였다. 중복된 이슈 없이 개발자가 동의한 결함 이슈 중, 유사한 원인으로 결함이 발생한 경우에 대해 분류하고 사례로 구분하여 수집하였다. 결함이 되는 원인은 이슈 내용에서 일부 확인 가능할 수 있다. 하지만 조사 과정 가운데, 이슈를 제기하고 이슈에 관해 설명하는 내용은 원인보다 결함으로 나타나는 증상에 집중되어

표 3 1차, 2차 분류 이후 필터링된 이슈의 개수

Table 3 The number of issues after the first and the second filtering processes

No.	project name	# of closed issues	# of issues (after the first filtering)	# of issues (after the second filtering)
1	DeepSpeech	1,905	98	86
2	OpenPose	1,559	98	79
3	spaCy	4,598	673	392
4	Real-Time-Voice-Cloning	646	4	3
5	tesseract	1,817	110	67
6	yolov5	3,207	436	143
7	mediapipe	2,159	50	16
8	ChatterBot	1,240	63	38
9	photoprism	833	154	123
10	streamlit	1,393	761	258
Total		13,189	2,447	1,205

있다는 것을 확인하였다. 따라서, 결함이 보고된 이슈의 내용뿐만 아니라 결함 이슈를 해결한 코드 수정을 확인해 명확하게 결함의 원인을 이해하고자 했다. 또한, 본 연구에서는 관련 연구에서 진행한 결함 분류를 이용하지 않고 애플리케이션의 이슈 리포트 내용을 바탕으로 새로운 분류를 진행했다. 관련 연구에서는 주로 기계학습 라이브러리에서 발생한 결함에 대해 분류했지만, 본 연구에서는 기계학습을 기반으로 구현된 애플리케이션에서 발생한 결함에 대해 유형을 나누어 정리하였다. 이는 조사 대상의 차이로 인해 결함 유발 원인 유형의 차이가 발생하고, 관련 연구의 제한적인 분류의 한계로 인해 기계학습 애플리케이션에서 발생하는 결함이 적절히 분류될 수 없었기 때문이다.

3.2.2.4 분류 과정

본 조사의 목적이 빈번하게 발생하는 결함의 원인에 대해 분석하는 것이기 때문에, 결함이 유발한 증상에 집중하기보다 결함이 유발된 근본적인 원인에 대해 조사하고자 하였다. 또한, 코드 수준에서의 상세한 유발 원인으로는 일반성을 지닌 범주로 결함 유발 사례들을 분류하기 어렵다는 문제점을 해결하기 위해, 상세한 코드 수준의 원인이 아닌 추상적인 수준에서의 원인을 조사하여 분류하였다. 예를 들면, 그림 2는 '예외적인 데이터 처리 누락'을 원인으로 분류한 결함 코드 수정 사례이다. 파이썬에서 제공하는 all 키워드 대신 any라는 키워드를 사용함으로써 인자로 들어오는 값에 대해서 사이즈가 0인 Doc도 처리할 수 있도록 만들어 준 것이다. 코드 수준에서 원인을 찾는다면, '어떤 텐서의 사이즈를 확인하는 경우에, 텐서의 사이즈가 all 함수의 인자로 사용됨'과 같이 패턴화할 수 있겠지만, 각 프로젝트의 요구사항에 따라 결함이 될 수도, 결함이 되지 않을 수도 있다. 본 예시 코드의 사용 목적은, 모든 입력 데이터값의 크기가 0보다 큰 경우가 있는지 확인하고자 하였기 때문에, '예외적인 데이터 처리 누락'이라는 더욱더 높은 수준에서의 원인 분석을 통해 결함 분류의 일반성을 높이고자 하였다. 즉, 기계학습 프로젝트에서 더욱 주의를 기울여 신중하게 코드를 작성하여야 하는 부분에 대해 일반적인 관점에서 접근하여, 기계학습 코드를 작성하는 개발자들에게 가이드라인을 제시할 수 있는 결함의 원인을 중심으로 분류하였다.

spacy/pipeline/tok2vec.py

```
- if not all(doc.tensor.size for doc in inputs):
+ if not any(doc.tensor.size for doc in inputs):
```

그림 2 코드 수정 사례 예

Fig. 2 An example of a defect-fixing change

4. 연구 결과

4.1 결함 유발 코드 사례

4.1.1 결과 개요

조사 대상으로 선정한 10개의 오픈 소스 프로젝트 중 5개 이상의 프로젝트에서 발견할 수 있는 이슈를 대표적 결함 원인으로 선정하였다. 표 4는 3.2.2에서 명시한 세 단계에 걸쳐 수집된 대표적인 결함의 원인 목록이다. 표 4의 빈도 열은 1,205개의 이슈 중 각 원인이 발견된 이슈의 개수를 의미한다. 표 4에 제시된 있는 순번은 빈도를 기준으로 내림차순으로 정렬한 결과이다. 가장 높은 빈도로 나타난 결함 유발 원인은 '예외적인 데이터에 대한 처리 누락'으로, 발견된 결함 1,205개 중 214개의 사례가 발견되었다. 이슈가 보고된 프로젝트의 개수가 가장 많은 원인은, '기본값/옵션값의 잘못된 설정 및 누락', '라이브러리 임포트/설치 문제', '오타'로 총 10개의 프로젝트 중 9개의 프로젝트에서 발견되었다. 5개 미만의 프로젝트에서 발견된 결함 유발 원인의 경우, 일반성이 떨어지는 한계가 있는 것으로 상정하고 본 논문에 보고하지 않았다. 하지만, 발견한 모든 결함 유발 원인이 다양한 연구에 사용될 수 있기 때문에 추가로 웹 페이지에 게시하였다.[1)]

표 4에 나타난 결과와 같이, 총 20개의 결함 유발 원인 중, 6개의 결함 원인(No. 1, 3, 4, 7, 11)이 데이터 처리와 연관되어 있음을 알 수 있다. 이는 기계학습 애플리케이션이 기계학습을 기반으로 한 프로그래밍 패러다임, 즉, 방대한 양의 데이터를 바탕으로 모델을 학습시켜 문제를 해결하는 방식을 따르기 때문으로 판단할 수 있다. 특별히, 존재하는 방대한 데이터의 여러 형태를 고려하지 못해 발생하는 결함, 데이터를 학습하는 과정 가운데에 데이터의 누락을 원인으로 발생하는 결함, 여러 데이터를 다루는 과정 가운데 데이터의 타입을 알맞게 설정해 처리하지 않은 경우, 데이터와 데이터 사이에서 잘못된 정렬, 배치, 비교 등 직접적으로 데이터를 처리하면서 발생할 수 있는 여러 예외를 고려하지 못한 사례들이 높은 비중을 차지한다. 또한, 많은 양의 데이터를 수집하는 과정에서 데이터를 읽어오는 웹 주소의 설정이 잘못된 경우, 혹은 수집한 데이터를 저장한 디렉터리의 경로를 잘못 설정하는 경우 등, 데이터를 읽어오는 과정에서 유효하지 않은 설정이 결함의 원인이 된 경우도 여러 프로젝트에 걸쳐 발생했다는 사실을 발견할 수 있었다.

4.1.2 일반적인 애플리케이션의 결함 사례와의 비교

4.1.1에서 정리한 연구 결과를 바탕으로, 조사된 결함 사례들이 기계학습 애플리케이션 뿐만 아니라, 다양한 일반적인 애플리케이션에서도 발견됨을 확인했다. 따라

1) https://bit.ly/3GWvDVi

표 4 결함 유발 코드 사례 목록
Table 4 Observed defects in the open source artificial intelligence applications

No.	Cause	frequency	# of projects
1	Missing exceptional data handling	214	8
2	Invalid device ordinal/Incorrect implementation for devices/OS	84	8
3	Incorrectly preserving previous data with all attributes	81	8
4	Missing necessary attributes of data/environment	60	7
5	Typo	52	9
6	Missing/Incorrect Error Handling	45	7
7	Improper data type/data source/operations for processing	44	6
8	Broken consistency between modules of the model	39	7
9	Missing/incorrect default option/value setting for the system	35	9
10	Documentation/manual error	35	8
11	Incorrectly augmenting/sorting/comparison/alignment for data/batch size	32	6
12	External library version issue	32	8
13	Unable to override/configure new options into default configurations	28	7
14	Missing or incorrect process for exceptional environment	28	6
15	Missing/incorrect import/installation	27	9
16	Incorrectly saving/caching trained model/Incorrectly print the output/result	15	5
17	Incorrect URL/data loading/path when crawling/processing	14	7
18	Memory Leak/Improper memory management	12	5
19	Incorrectly manage multiprocessing environment/race condition	9	6
20	Improper encoding/decoding	9	5

서, 과거의 일반적인 애플리케이션의 결함 사례 분석 관련 연구 결과와 본 연구의 조사 결과를 비교하였다. Li et al.은 코드 수준에서 기능적인 측면을 중심으로 Mozilla, apache, Linux kernel에서 수집한 2,060개의 결함 사례를 세 가지 분류로 구분했다[6]. 한편, Catolino et al.은 기능적인 결함 사례 뿐 아니라, 설정 관련 문제(configuration issue), 통신망 관련 문제(network issue), 테스트 코드 문제(test code-related issue) 등 소프트웨어 전반적인 수준에서 발생한 결함에 대해 조사했다. 해당 연구에서는 Mozilla, Apache, Eclipse에 속하는 119개의 프로젝트에서 'fixed'와 'closed' 라벨을 가진 결함 보고 중 무작위로 1,280개의 결함을 수집하여 분석을 진행했다[22].

Li et al.의 연구에서 일반적인 애플리케이션의 결함 유발 원인을 분석한 결과, 잘못된 기능 구현(wrong functionality implementation), 누락된 기능(missing feature), 포괄적 의미의 잘못된 처리(processing), 잘못된 예외 처리(exception handling) 등이 주된 결함 유발 원인이 됨을 확인하였다[6]. 잘못된 기능 구현은 전체 사례 중 35.9%로 가장 많은 결함 사례들을 포함한다. 본 연구에서 해당 범주에 포함되는 결함 유발 원인은 표 4의 No. 2와 No. 19에 해당하는 범주로, 전체 결함 유발 원인의 10.4%를 차지함을 확인할 수 있다. 다음으로, 높은 비율을 차지하는 결함 원인은 누락된 기능이며 7%의 비율을 차지한다. 해당 범주에는 No. 8과 No. 9를 포함할 수 있으며, 약 8.3%의 비율을 차지한다. 한편, 포괄적 의미의 잘못된 처리는 일반적인 애플리케이션에서 발견한 결함 사례 중 3.2%의 비율을 차지했으며, No. 3과 7, 11, 16, 17, 20, 총 6개 범주가 포괄적 의미의 처리에 해당한다. 이는 총합 21.8%로 기계학습 애플리케이션에서 특징적으로 높은 비율을 가진다. 마지막으로, 예외 처리 범주의 경우, 일반적인 애플리케이션의 결함 사례에서는 2.8%의 결함 사례를 포함했지만, 기계학습 애플리케이션의 결함 사례에서는 No. 1과 No. 6의 범주를 포함해 전체 결함 사례 중 27%의 비율 차지함을 확인할 수 있다.

Catolino et al.의 연구에서 분류한 결함 사례의 범주 중, 프로그램 이상 작동 관련 문제가 47%로 가장 빈번하게 발생 하였고 그 뒤를 이어 설정 관련 문제가 16%로 빈번하게 발생하였다[22]. 7%인 시험 코드 관련 문제를 제외한 나머지 범주들은 모두 4% 미만으로 낮은 비율을 차지했다. 가장 높은 비율을 차지한 두 가지의 범주를 중점적으로 살펴보면, 설정 관련 문제에 해당하는 결함 유발 원인은 No. 2, 3, 8, 9, 11, 12, 15, 17을 포함한 7개 범주가 있다. 이는 전체 결함 사례 중 총합 36%의 결함 사례를 포함된다. 반면에, 프로그램 이상 작동 관련 문제의 경우, No. 1, 6, 12, 15, 16, 18, 19, 20의 범주가 포함된다. 이는 전체 결함 사례 중 40%의

비율을 차지한다. 해당 연구에서는 예외 처리 관련 문제와 프로그램 논리 문제로 인한 모든 충돌(crash) 오류를 해당 범주에 포함하기 때문에 본고의 연구 결과에서도 다소 포괄적인 결함 유발 사례들이 포함되어 두 연구 결과가 비슷한 수준의 비율을 보인다.

비교의 결과로 나타난 차이는 전통적인 프로그래밍 패러다임과 기계학습 프로그래밍 패러다임의 특징에 기인한다. 전통적인 프로그래밍 패러다임에서는 문제 해결에 적합한 알고리즘을 통해 직접 기능을 구현하기 때문에, 각 기능에 대한 구현의 중요도가 높다. 하지만, 데이터를 기반으로 문제를 해결하는 기계학습 패러다임에서는 기계학습 모델을 설계하고 설정하는 과정과 데이터를 처리하고 학습하는 과정이 더욱 중요하기 때문에, 잘못된 기능 구현 관련 결함이 차지하는 비율이 일반적인 애플리케이션보다 상대적으로 낮고 설정 관련 결함이 차지하는 비율이 높은 것으로 해석할 수 있다. 특히, 기계학습 애플리케이션에서 잘못된 설정이 결함의 원인으로 파악되는 사례는 Catolino et al.의 연구에서 조사된 비율의 2배 이상으로, 설정 관련 문제가 기계학습 애플리케이션에서 더욱 빈번하게 발생하는 것을 알 수 있다. 마지막으로 기계학습 애플리케이션의 결함 유발 사례에서 예외 처리 관련 결함이 일반적인 소프트웨어보다 빈번하게 발생하는 것을 확인할 수 있다. Li. et al.의 연구와 본 연구의 비교를 통해, 전체 결함 사례 중 잘못된 예외 처리로 발생한 결함 사례의 비율이 일반적인 애플리케이션보다 약 10배가량 높은 비중을 차지함을 확인할 수 있다. 이와 같은 이유는 단순히 기계학습 애플리케이션에서 일어날 수 있는 예외 상황이 많은 것으로 설명될 수 있지만, 기계학습 프로그래밍 패러다임에서 일어날 수 있는 예외 상황들과 이들의 처리에 대한 일반화가 전통적인 방식이랑 비교했을 때 부족하기 때문으로도 설명될 수 있다.

4.1.3. 대표 사례

본 연구에서 관찰한 대표적인 결함 유발 원인의 코드 수정 사례는 아래와 같다.

4.1.3.1 예외적인 데이터에 대한 처리 누락

데이터를 읽어오고 처리하는 과정에서 발생할 수 있는 예외적인 데이터 값에 대한 처리를 누락한 경우로, 예외적인 데이터 값에 대한 crash가 발생하거나 모델 성능에 부정적인 영향을 줄 수 있다. 그림 3은 해당 사례의 예시이다. 본 예시에서는 텐서의 사이즈를 확인할 때, 텐서의 사이즈를 all 함수의 인자로 처리하여 입력값 중 사이즈가 0인 값이 하나라도 존재하는 경우, 의도된 동작을 할 수 없었던 문제가 있었다. all 함수 대신 any 함수를 사용해서 예외적인 상황에서도 데이터를 적절하게 처리할 수 있도록 해결하였다.

```
spacy/pipeline/tok2vec.py
-  if not all(doc.tensor.size for in inputs);
+  if not any(doc.tensor.size for in inputs);
```

그림 3 예외적인 데이터에 대한 처리 누락 사례

Fig. 3 Missing exceptional data handling

4.1.3.2 특정 하드웨어/장치/운영 체제에 대한 잘못된 구현

특정 GPU, CPU 환경에 대해 각각의 경우를 고려하지 않고 구현해 하드웨어 및 장치의 효율을 극대화하지 못하거나 부적절한 처리를 해 crash가 발생하는 경우가 있다. 또한, 윈도우, 리눅스, 맥 등의 특정 운영 체제에 프로젝트가 사용될 때 파일 경로 구분자 문제 등으로 구현이 특정 환경에서 빌드 실패되거나 crash가 발생하는 사례도 본 분류에 추가하였다. 그림 4는 해당 사례의 예시이다. 본 예시에서 run-ldc93s1 스크립트는 하나의 GPU에서만 동작할 수 있으며 두 개 이상의 GPU에서 실행시킬 경우 오류가 발생하였다. 스크립트 내부에서, GPU 환경에 대한 처리를 누락하였으며, 강제로 GPU를 하나만 사용하는 코드를 추가하여 문제를 해결하였다.

```
src/textord/colfind.cpp
-  column_sets = &temp_cols;
+  *columns_sets = temp_cols;
```

그림 4 특정 하드웨어/장치/운영 체제에 대한 잘못된 구현 사례

Fig. 4 Invalid device ordinal/incorrectimplementation for devices/OS

4.1.3.3 데이터 처리 과정 중 이전 데이터 누락

이전의 데이터가 새로운 데이터에 의해 덮여 씌워지거나, 데이터의 누락 등으로 인해 가지고 있던 온전한 데이터가 보존되지 못하는 경우로, 데이터는 학습에 필요한 데이터, 평가에 필요한 데이터, 값 처리에 대한 데이터를 모두 포함한다. 성능 저하, 의도하지 않은 동작, crash를 야기한다. 그림 5는 해당 사례의 예시이다. 본 예시에서 pop 함수를 사용할 경우, 데이터에 대한 보존 없이 가지고 있던 데이터를 리스트에서 삭제하게 된다. 이 경우, 임포트해야할 경로에 대한 누락으로 인해 crash가 발생하였고, 리스트 원소 삭제 없이 값을 가져오는 get 함수로 수정함으로써 해결하였다.

```
chatterbot/utils.py
-  import_path = data.pop('import_path')
+  import_path = data.get('import_path')
```

그림 5 데이터 처리 과정 중 이전 데이터 누락

Fig. 5 Incorrectly preserving previous data with all attributes

4.1.3.4 필요한 데이터 속성(attributes) 누락

데이터 속성을 누락한 상황에서 모델을 학습하게 되

면, 학습에 필요한 데이터 속성이 없어서 crash가 발생하거나 데이터의 속성값이 누락되어 데이터가 정확히 처리되지 못하는 사례이다. 그림 6에서는 해당 사례를 보여준다. 데이터를 직렬화하는 경우에, 데이터에 대한 모든 속성이 포함되어야 한다. 하지만, token.norm_, token.ent_id_ 속성이 누락되어 있었고 누락된 속성을 추가함으로 문제를 해결하였다.

```
spacy/tokens/_serialize.py
+  self.strings.add(token.norm_)
+  self.strings.add(token.ent_id_)
```

그림 6 필요한 데이터 속성(attributes) 누락 사례

Fig. 6 Missing necessary attributes of data/ environment

4.1.3.5 오타

개발자의 실수로 코드 작성에 실수가 있는 사례로, 함수 호출 시 전달 인자의 순서를 잘못 사용하거나, 리스트 인덱스값을 잘못 작성, 반환 값의 개수 처리 실수 등 코드에 오류가 있는 경우이다. 오동작, crash를 야기한다. 그림 7은 해당 사례의 예시이다. 해당 예시는 함수를 통해 반환되는 값의 개수에 대해 정확하게 처리하지 못해 발생한 문제로 실행 시간에 오류를 일으키며 crash가 발생하였다. 모든 반환되는 값에 대해 처리해줌으로써 해결하였다.

```
synthesizer/synthesize.py
-   _, mels_out, _ = model(texts, mels, embeds)
+   _, mels_out, _, _ = model(textx, mels, embeds)
```

그림 7 오타 사례

Fig. 7 Typo

4.1.3.6 잘못된 오류 핸들링

소프트웨어가 동작하면서 오류에 대해 정확한 메시지를 출력하고 의도한 대로 오류를 핸들링해야 하지만, 잘못된 오류 메시지를 출력하거나 오류 핸들링 과정이 누락되어 있는 사례로 crash를 야기한다. 그림 8에서는 해당 사례를 보여준다. 오류가 발생했을 때, 따로 그 오류에 대해 처리하는 메커니즘이 없어 정확한 핸들링을 하지 못하여 crash가 발생한 사례로, 오류에 대한 핸들링 과정을 추가함으로써 해결하였다.

```
internal/photoprism/metadata.go
+  “errors”
-   err:= m.metaData.Exif(m.FileName())
+  var err error
+  if m.IsPhoto() {
+     err = m.metaData.Exif(m.FileName())
+  } else {
+     err = errors.New(“Not a photo”)
+  }
```

그림 8 잘못된 오류 핸들링 사례

Fig. 8 Missing/incorrect error handling

4.1.3.7 처리에 알맞지 않은 데이터/데이터 타입/연산자 사용

필요한 연산에 알맞지 않은 데이터/데이터 타입/연산자를 사용할 경우, 성능 평가 및 학습 성능에 대한 문제를 일으키거나 crash를 일으키거나 오동작할 수 있다. 아래 그림 9의 예시는 포인터를 사용해 저장해야 할 데이터를 단순 값으로 저장해 애플리케이션이 오동작한 사례이다.

```
bin/run-idc93s1.sh
+  #  Force only one visible device because
+  #  we have a single-sample dataset
+  #  and when trying to run on multiple
+  #  devices (like GPUs), this will break
+  export CUDA_VISIBLE_DEVICES=0
```

그림 9 처리에 알맞지 않은 데이터/데이터 타입/연산자 사용 사례

Fig. 9 Improper data type/data source/operations for processing

4.1.3.8 모듈 간의 일관성 유지 실패

유사한 역할을 하는 모듈이 서로 다른 방식으로 데이터를 처리하거나 입력받거나 모델을 학습시켜 모듈 간의 일관성 유지 실패, 혹은 삭제된 모듈의 API를 계속해 사용해 빌드 실패/컴파일 오류/crash 등을 야기하는 사례이다. 그림 10에서는 해당 사례를 보여준다. 사용자가 설정한 sents값을 사용하고자 할 때, sents에 대한 속성과 doc.pyx 파일에서 처리하려고 한 방식이 맞지 않아 발생한 문제이다. sents는 생성자로 빈값을 반환하게 되므로, sent 클래스의 속한 값을 반복하여 확인하고 값을 생성해 반환함으로써 문제를 해결했다.

```
spacy/tokens/doc.pyx
-  return self.user_hooks[‘sents’](self)
+  for sent in self.user_hooks[‘sents’](self)
+     yield sent
+  return
```

그림 10 모듈 간의 일관성 유지 실패 사례

Fig. 10 Broken consistency between modules of the model

4.1.3.9 기본값/옵션값의 잘못된 설정 및 누락

애플리케이션에서 제공하는 옵션에 대한 분기 처리가 누락되거나 잘못 설정된 경우, 혹은 필요한 기본값이 누락되어 있거나 잘못 설정된 경우의 결함을 해당 사례 분류에 포함하였다. 그림 11은 exclude에 대한 기본값이 정확하게 설정되어 있지 않아서 오동작한 경우로, exclude에 대한 기본값을 설정해서 해결하였다.

4.1.3.10 문서/매뉴얼 오류

애플리케이션을 실행하거나 빌드하는 문서/매뉴얼에 잘못된 설명이 있거나 설명이 누락된 부분이 있는 경우

```
spacy/util.py
-   return nlp.from_disk(model_path)
+   return nlp.from_disk(model_path, exclude=disable)
```

그림 11 기본값/옵션값의 잘못된 설정 및 누락 사례
Fig. 11 Missing/incorrect default option/value setting for the system

```
docs/solutions/selfie_segmentation.md
+   import numpy as np
```

그림 12 문서/매뉴얼 오류 사례
Fig. 12 Documentation/manual error

에 사용자가 해당 애플리케이션을 정확하게 사용하거나 예시 코드의 동작을 확인할 수 없다. 그림 12에서는 해당 사례를 보여준다. 애플리케이션 예시 코드에서, numpy를 임포트하는 코드가 문서에 누락되어 해당 부분을 추가하여 해결하였다.

4.1.3.11 데이터의 잘못된 비교, 정렬, 배치, 통합

데이터들을 처리해주는 방식에 있어서 비교, 정렬, 배치할 경우, 정확한 비교를 하지 않거나, 잘못된 정렬을 하거나, 배치 사이즈를 잘못 설정하거나 통합하여 정확한 데이터 처리가 불가한 경우로, crash를 발생시키거나 성능 저하, 오동작으로 이어진다. 그림 13에서는 해당 사례를 보여준다. stripSequence 값에 따라 다른 방식을 통해 데이터를 비교/검색하기 위한 정규식을 구성해야 하는데, 모든 데이터에 같은 방식의 정규식을 만들어 데이터 비교/검색에 문제가 발생한 경우이다. 분기 문을 통해 적합한 정규식을 만들어 데이터 비교/검색에 문제를 해결하였다.

```
internal/photoprism/mediafile.go
-   matches, err:= filepath.Glob(regexp.QuoteMeta(
m.AbsPrefix(stripSequence)) + "*")
+   var prefix string
+   if stripSequence {
+       prefix = regexp.QuoteMeta(m.AbsPrefix(true))
+   } else {
+       prefix = regexp.QuoteMeta(m.AbsPrefix(false) + ".")
+   }
+   matches, err := filepath.Glob(prefix + "*")
```

그림 13 데이터의 잘못된 비교, 정렬, 배치, 통합 사례
Fig. 13 Incorrectly augmenting/sorting/comparison/alignment for data/batch size

4.1.3.12 외부 라이브러리 버전 문제

빌드 과정에서 필요한 라이브러리의 버전이 명시되지 않은 경우, 사용자의 실행 환경에 따라 각각 다른 버전의 외부 라이브러리를 사용할 수 있는데, 이는 빌드 실패로 이어지게 된다. 따라서, 명시적으로 어떤 버전의 외부 라이브러리를 사용해야 하는지 확인해주어야 한다. 최신 버전을 사용하지 않아 발생하는 문제가 다수 발생했고, 버전 업그레이드를 통해 해결하였다. 또한 외부 라이브러리의 버전이 업데이트되며 안정화되지 않은 요소를 사용할 경우, 버그가 발생할 수 있다. 이를 방지하기 위해 최신 버전을 사용하기보다 의도적으로 낮은 버전의 외부 라이브러리를 사용한다. 그림 14 사례의 경우, 외부 라이브러리인 15_amd64의 버전을 16_amd64로 변경하여 문제를 해결하였다.

```
taskcluster/tc-py-utils.sh
-   http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl-dev_1.0.2g-
1ubuntu4.15_amd64.deb /
-   http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl1.0.0_1.0.2g-
1ubuntu4.15_amd64.deb
+   http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl-dev_1.0.2g-
1ubuntu4.16_amd64.deb /
+   http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl1.0.0_1.0.2g-
1ubuntu4.16_amd64.deb
```

그림 14 외부 라이브러리 버전 문제 사례
Fig. 14 External library version issue

4.1.3.13 사용자 옵션/설정에 대한 업데이트 불가

사용자의 옵션 및 설정에 따라 알맞게 값이 변경되어야 하는데, 업데이트가 불가능한 상수값으로 기본값을 설정한 경우, 혹은 업데이트되어야 하는 변수가 정의되어 있지 않은 경우, 의도하지 않은 동작을 하거나 성능 저하, crash가 발생한다. 그림 15에서는 해당 사례를 보여준다. 픽셀의 해상도 기본값을 pixSetYRes 함수를 사용해 설정하였지만, 이 함수는 const 타입을 사용해 사용자의 설정/옵션에 따라 업데이트될 수 없게 코드가 작성되어 성능 저하를 야기했다. 픽셀의 해상도 기본값에 대해 설정하는 코드를 삭제함으로 해결하였다.

```
src/ccmain/thresholder.cpp
-   pixSetYRes(pix, 300);
```

그림 15 사용자 옵션/설정에 대한 업데이트 불가 사례
Fig. 15 Unable to override/configure new options into default configurations

4.1.3.14 예외적인 실행 환경에 대한 처리 누락

불안정한 네트워크 연결 상태 등의 발생 가능한 실행 환경의 문제에 대해 추가로 처리하는 부분이 누락되거나 충분하게 처리하지 못해 발생하는 문제로, 의도하지 않은 동작을 하거나 성능 저하, crash가 발생한다. 그림 16에서는 해당 사례를 보여준다. pip로 라이브러리 관리를 하는 프로젝트로, pip 시간 초과 환경 변숫값을 예외적인 상황에 대한 고려 없이 기본값으로 설정하여 발생한 문제로, 시간 초과 환경 변숫값을 기본값보다 더 큰 값으로 설정해 문제를 해결하였다.

4.1.3.15 라이브러리 임포트/설치 문제

필요한 라이브러리를 임포트하지 않거나 임포트의 범

taskcluster/test-darwin-opt-base.tyml

```
+ export PIP_DEFAULT_TIMEOUT=60 &&
```

그림 16 예외적인 실행 환경에 대한 처리 누락 사례
Fig. 16 Missing or incorrect process for exceptional environment

위가 잘못 설정되어 crash가 발생하는 경우, 혹은 설치해야 하는 라이브러리를 빌드 파일에서 명시하지 않아 빌드 실패가 되는 사례이다. 그림 17에서는 해당 사례를 보여준다. 임포트하는 라이브러리를 잘못 설정하여 발생한 문제로 적절한 라이브러리로 교체하여 문제를 해결하였다.

data/scripts/download_weights.sh

```
- from utils.google_utils import attempt_download
+ from utils.downloads import attempt_download
```

그림 17 라이브러리 임포트/설치 문제 사례
Fig. 17 Missing/incorrect import/installation

4.1.3.16 잘못 저장된 모델/캐싱 혹은 잘못된 출력값

학습된 모델을 재사용하기 위해 파라미터 등을 유지하여 모델을 저장해야 하는데, 필요한 모델의 속성이 충분히 고려되지 않고 학습된 모델이 아닌 기본 모델을 저장하거나 학습이 제대로 반영되지 않은 채로 모델을 저장하는 사례다. 그림 18에서는 해당 사례를 보여준다. 학습된 모델을 저장하고 출력하면서, pred 값이 누락되어 생성된 모델이 정확히 저장되지 않는 경우로, 변수 x를 통해 누락되는 값이 없도록 저장함으로써 해결하였다.

test.py

```
- txt_path = scr(out / Path(paths [si].)stem)
- pred[:, :4] = scale_coords(img[si].shape[1:], pred[:, :4], shapes[si] [0], shapes [si] [1])
- for *xyxy, conf, cls in pred:
+ x = pred.clone()
+ x[:, :4] = scale_coords(img[si].shape[1:], x[:, :4], shapes[si] [0], shapes [si] [1])
+ for *xyxy, conf, cls in x:
-     with open(txt_path + '.txt', 'a') as f:
-     with open(str(out / Path(paths[si].stem) + '.txt', 'a') as f:
```

그림 18 잘못 저장된 모델/캐싱 혹은 잘못 출력된 값 사례
Fig. 18 Incorrectly saving/caching trained model/ Incorrectly print the output/result

4.1.3.17 데이터 처리를 위한 경로/웹 주소의 잘못된 설정

데이터 로딩/크롤링을 하기 위한 서버/데이터베이스 주소가 잘못 설정되어 있거나, 실행 예시에서 데이터를 다운받는 경로와 데이터를 불러오는 경로가 다른 경우, crash나 오동작을 야기한다. 그림 19에서는 해당 사례를 보여준다. 애플리케이션에서 제공하는 예시 코드 중 데이터베이스에 접근하는 URL이 잘못 설정되어 있어서 발생한 문제로, 잘못 설정한 경로에 대해 수정함으로써 수정하였다.

examples/terminal_example.py

```
- database_url = "../database.db"
+ database_url = "sqlite:///database.db"
```

그림 19 데이터 처리를 위한 경로/웹 주소의 잘못된 설정 사례
Fig. 19 Incorrect URL/data loading/path when crawling/ processing

4.1.3.18 메모리 누수/ 잘못된 메모리 관리

메모리 할당 이후 해제하지 않는 경우, 혹은 잘못된 메모리 관리로 out-of-memory로 인한 crash가 발생하는 경우다. 그림 20에서는 해당 사례를 보여준다. 할당된 activations를 메모리로부터 해제하는 코드가 누락되어 발생한 메모리 누수로, free 함수를 구현 및 사용함으로써 해결하였다.

spacy/syntax/_parser_model.pyx

```
+ cdef activationsC alloc_activations(SizeC n) nogil:
+         cdef ActivationsC A
+         memset (&A, 0, sizeof(A))
+         resize_activations(&A, n)
+         return A
+
+
+ cdef void free_activations(const ActivationsC* A) nogil:
+         free(A.token_ids)
+         free(A.scores)
+         free(A.unmaxed)
+         free(A.hiddens)
+         free(A.is_valid)
```

그림 20 메모리 누수/잘못된 메모리 관리 사례
Fig. 20 Memory leak/impropermemory management

4.1.3.19 잘못된 병렬화 구현

소프트웨어 병렬화를 구현할 때 소프트웨어 상태를 명확히 정의하고 병렬화 과정 가운데 서로 다른 스레드(thread)/프로세스의 불필요한 간섭이 제한되어야 하는데, 이를 충분히 고려하지 못한 사례이다. 멀티 스레드를 사용할 때, 각 스레드의 상태 값을 확인하지 않아 발생한 문제로, 스레드 상태 값을 확인하는 코드를 추가하여 수정하였다.

4.1.3.20 잘못된 인코딩/디코딩

데이터 인코딩 및 디코딩 과정에서 부적합한 인코딩 방식을 사용한 사례이다. 그림 21에서는 해당 사례를 보여준다. 인코딩 방식을 명시하지 않아 발생한 문제로, 사용할 인코딩 방식을 명시함으로써 해결하였다.

spacy/cli/evaluate.py

```
-     with (output_path / "entitles.html").open("w") as file_;
+     with (output_path / "entities.html").open("w", endodinf="utf8") as file_;
```

그림 21 잘못된 인코딩/디코딩 사례
Fig. 21 Improper encoding/decoding

5. 타당도 위협 요인

조사 대상으로 선정한 오픈소스 애플리케이션의 결함 보고 이력과 코드 수정 이력을 기반으로 각 저자가 직접 각각의 결함 사례를 나누어 분석하였다. 따라서, 저자의 주관이 개입되어 분류에 일관성을 해치는 내적 타당도(internal validity) 문제가 발생할 수 있다. 이처럼, 발생할 수 있는 내적 타당도를 위협하는 요인을 완화하기 위해, 각 저자가 분석하고 분류한 결함 원인과 증상을 서로 확인하고 검증하는, 교차 검증 절차를 거쳤다. 한편, 모든 이슈 보고를 파악하지 않고 필터링 단계를 거쳐 결함 보고 사례 중 결함과 관련이 있는 라벨이 붙어 관리되고 있는 이슈만을 중심으로 분석하였다는 점에서도 내적 타당도의 문제가 있을 수 있다.

6. 결론 및 향후 연구

본 연구에서는, 기계학습 기법을 사용하는 다양한 오픈소스 애플리케이션에서 발견한 결함을 기반으로 기계학습 기반 애플리케이션에서 나타날 수 있는 결함의 원인과 증상을 분석하였다. 결함을 분석하기 위해 GitHub 애플리케이션 10개에 보고된 1,205개의 결함 보고를 직접 분석하였으며, 연구 결과 20개의 결함 원인을 정리하였다. 가장 높은 빈도로 나타난 결함 유발 원인은 '예외적인 데이터에 대한 처리 누락'이었으며, 전체 결함 중 약 18%의 비중을 차지하였다. 이슈가 보고된 프로젝트의 개수가 가장 많은 원인은, '기본값/옵션값의 잘못된 설정 및 누락', '라이브러리 임포트/설치 문제', '오타'로, 총 10개의 프로젝트 중 9개의 프로젝트에서 발견되었다. 또한, 발견한 결함 유발 원인이 주로 데이터 처리와 연관됨을 확인하였다. 이는 일반적인 애플리케이션에서 주로 발견할 수 있는 기능 구현 관련 결함 유발 원인의 특징과는 상이하다. 이와 같은 차이는 기계학습 프로그래밍 패러다임에서 기계학습 모델을 설계하고 설정하는 과정과 데이터를 처리하고 학습하는 과정이 더욱 강조된다는 것으로 설명할 수 있다.

본 연구의 결과를 바탕으로 결함 유발 원인이 되는 기계학습 애플리케이션의 코드를 자동으로 검출하는 결함 검출 도구를 구현할 수 있다. 각각의 결함 유발 사례의 결함 유발 코드의 패턴을 추출한다면, 기계학습 프로젝트에서 빈번하게 발생하는 결함을 자동으로 검출하는 규칙 기반 결함 검출 도구를 설계할 수 있다. 또한, 본 연구 결과를 기계학습을 이용한 소프트웨어 분석 및 결함 검출 기법의 훈련 데이터로 사용할 수 있다. 예를 들면, 본 연구 결과는 이슈를 기반으로 결함 유발 원인을 파악하고 분류했기 때문에, 정보 검색 기반 결함 위치 추적 기법을 설계하는 것에 활용될 수 있다. 또한, 결함을 해결한 코드를 학습해, 결함 해결을 위한 코드 수정을 제안하는 방식의 기법에도 활용할 수 있다.

발견한 결함 원인을 바탕으로, 빈번하게 발생하는 결함의 원인으로 작용한 코드에 대해 더욱 면밀하게 분석하여, 기존의 결함 검출 도구가 검출하지 않거나 검출하기 어려웠던 기계학습 기반 애플리케이션의 결함을 자동으로 검출하는 결함 검출 도구를 개발하는 방향으로 연구를 확장할 계획이다. 또한, 빈번하게 발생하는 결함의 코드를 해결한 코드 수정 사례를 수집하여 자동으로 결함이 있는 기계학습 기반 애플리케이션의 문제를 해결하는 방식으로 연구를 확대할 계획이다.

References

[1] McMahan, B. and Rao, D., "Listening to the world improves speech command recognition," *Proc. of the AAAI Conference on Artificial Intelligence*, Vol. 32, No. 1, Apr. 2018.

[2] Furuta, R., Inoue, N., and Yamasaki, T., "Fully convolutional network with multi-step reinforcement learning for image processing," *Proc. of the AAAI conference on artificial intelligence*, Vol. 33, No. 01, pp. 3598-3605, Jul. 2019.

[3] Chen, Z., Shen, S., Hu, Z., Lu, X., Mei, Q., and Liu, X., "Emoji-powered representation learning for cross-lingual sentiment classification," *The World Wide Web Conference*, pp. 251-262, May. 2019.

[4] Li, P., Chen, X., and Shen, S., "Stereo r-cnn based 3d object detection for autonomous driving," *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7644-7652, 2019.

[5] Zhang, Y., Chen, Y., Cheung, S. C., Xiong, Y., and Zhang, L., "An empirical study on TensorFlow program bugs," *Proc. of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 129-140, Jul. 2018.

[6] Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., and Zhai, C., "Have things changed now? An empirical study of bug characteristics in modern open source software," *Proc. of the 1st workshop on Architectural and system support for improving software dependability*, pp. 25-33, Oct. 2006.

[7] Lu, S., Park, S., Seo, E., and Zhou, Y., "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics," *Proc. of the 13th international conference on Architectural support for programming languages and operating systems*, pp. 329-339, Mar. 2008.

[8] Pan, K., Kim, S., and Whitehead, E. J., "Toward an understanding of bug fix patterns," *Empirical Software Engineering*, Vol. 14, No. 3, pp. 286-315, 2009.

[9] Thung, F., Wang, S., Lo, D., and Jiang, L., "An empirical study of bugs in machine learning systems,"

2012 IEEE 23rd International Symposium on Software Reliability Engineering, pp. 271-280, IEEE, Nov. 2012.
[10] Zhang, R., Xiao, W., Zhang, H., Liu, Y., Lin, H., and Yang, M., "An empirical study on program failures of deep learning jobs," *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pp. 1159-1170, IEEE, Oct. 2020.
[11] Islam, M. J., Nguyen, G., Pan, R., and Rajan, H., "A comprehensive study on deep learning bug characteristics," *Proc. of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 510-520, Aug. 2019.
[12] Mozilla. (2016, Feb 22). DeepSpeech [Online]. Available: https://github.com/mozilla/DeepSpeech (Retrieved 2021, Dec, 4)
[13] CMU-Perceptual-Computing-Lab. (2017, Apr 24). openpose [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose (Retrieved 2021, Dec. 4)
[14] Explosion. (2014, Jul 4). spacy [Online]. Available: https://github.com/explosion/spaCy (Retrieved 2021, Dec. 4)
[15] CorentinJ. (2018, Sep 16). real-time-voice-cloning [Online]. Available: https://github.com/CorentinJ/Real-Time-Voice-Cloning (Retrieved 2021, December 4)
[16] Tesseract-OCR. (2007, Mar 8). tesseract [Online]. Available: https://github.com/tesseract-ocr/tesseract (Retrieved 2021, Dec. 4)
[17] Ultralytics. (2020, May 30). yolov5 [Online]. Available: https://github.com/ultralytics/yolov5 (Retrieved 2021, Dec. 4)
[18] Google. (2019, Jun 17). mediapipe [Online]. Available: https://github.com/google/mediapipe (Retrieved 2021, Dec. 4)
[19] Gunthercox. (2014, Sep 7). Chatterbot [Online]. Available: https://github.com/gunthercox/ChatterBot (Retrieved 2021, Dec. 4)
[20] Photoprism. (2018, Jan 27). Photoprism [Online]. Available: https://github.com/photoprism/photoprism (Retrieved 2021, Dec. 4)
[21] Streamlit. (2018, Jan 10). Streamlit [Online]. Available: https://github.com/streamlit/streamlit (Retrieved 2021, Dec. 4)
[22] Catolino, G., Palomba, F., Zaidman, A., and Ferrucci, F., "Not all bugs are the same: Understanding, characterizing, and classifying bug types," *Journal of Systems and Software*, Vol. 152, pp. 165-181, 2019.

최 윤 호

2021년 2월 한동대학교 전산전자공학부 (학사). 2021년~현재 한동대학교 정보통신공학과 석사 과정. 관심분야는 소프트웨어 공학, 자연어 처리

이 창 공

2019년~한동대학교 전산전자공학부. 관심분야는 소프트웨어 공학, 자동 프로그램 수정

남 재 창

2015년 홍콩과기대 컴퓨터공학 박사. 2015년 9월 University of Waterloo, 박사후 연구원. 2017년 10월 POSTECH 연구교수. 2018년 3월~현재 한동대학교 전산전자공학부 조교수. 관심분야는 Mining software repositories. Automated debugging, Software quality prediction