JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 28 No. 10, pp. 123-132, October 2023
https://doi.org/10.9708/jksci.2023.28.10.123

Implementation of Git's Commit Message Classification Model Using GPT-Linked Source Change Data

Ji-Hoon Choi*, Jae-Woong Kim**, Seong-Hyun Park*

*Ph. Student, Dept. of Computer Engineering, Kongju National University, Chungnam, Korea
**Professor, Dept. of Computer Science and Engineering, Kongju National University, Chungnam, Korea
*Ph. Student, Dept. of Computer Engineering, Kongju National University, Chungnam, Korea

[Abstract]

Git's commit messages manage the history of source changes during project progress or operation. By utilizing this historical data, project risks and project status can be identified, thereby reducing costs and improving time efficiency. A lot of research related to this is in progress, and among these research areas, there is research that classifies commit messages as a type of software maintenance. Among published studies, the maximum classification accuracy is reported to be 95%. In this paper, we began research with the purpose of utilizing solutions using the commit classification model, and conducted research to remove the limitation that the model with the highest accuracy among existing studies can only be applied to programs written in the JAVA language. To this end, we designed and implemented an additional step to standardize source change data into natural language using GPT. This text explains the process of extracting commit messages and source change data from Git, standardizing the source change data with GPT, and the learning process using the DistilBERT model. As a result of verification, an accuracy of 91% was measured. The proposed model was implemented and verified to ensure accuracy and to be able to classify without being dependent on a specific program. In the future, we plan to study a classification model using Bard and a management tool model helpful to the project using the proposed classification model.

▸ **Key words**: Commit Message, GPT, Source Change, DistilBERT, Multi-Label Classification

• First Author: Ji-Hoon Choi, Corresponding Author: Jae-Woong Kim
*Ji-Hoon Choi (hunnx27@gmail.com), Dept. of Computer Engineering, Kongju National University
**Jae-Woong Kim (jykim@kongju.ac.kr), Dept. of Computer Science and Engineering, Kongju National University
*Seong-Hyun Park (a94270816@gmail.com), Dept. of Computer Engineering, Kongju National University
• Received: 2023. 09. 01, Revised: 2023. 10. 11, Accepted: 2023. 10. 12.

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

[요 약]

Git의 커밋 메시지는 프로젝트 진행 혹은 운영 과정에서 소스가 변경되는 이력을 관리한다. 이러한 이력 데이터를 활용하면 프로젝트 리스크와 프로젝트 현황을 파악할 수 있어 비용 절감과 시간 효율 개선을 높일 수 있다. 이와 관련된 많은 연구가 진행되고 있고 이러한 연구 분야 중 커밋 메시지를 소프트웨어 유지관리의 유형으로 분류하는 연구가 있다. 발표된 연구 중 최대 분류 정확도는 95%로 보고되어 있다. 본 논문에서는 커밋 분류 모델을 이용한 솔루션 등의 활용을 목적으로 연구를 시작했고, 기존 연구 중 정확도가 가장 높은 모델이 JAVA 언어로 작성된 프로그램에만 적용할 수 있는 제약을 없애기 위한 연구를 수행하였다. 이를 위해 GPT를 이용해서 소스 변경 데이터를 자연어로 표준화하는 단계를 추가 설계하고 구현하였다. 본문은 Git에서 커밋 메시지와 소스 변경 데이터를 추출하고, GPT로 소스 변경 데이터를 표준화하는 과정과 디스틸버트(DistilBERT) 모델을 이용한 학습 과정을 설명한다. 검증 결과 91%의 정확도를 측정하였다. 제안하는 모델은 정확도를 확보하고 특정 프로그램에 종속되지 않고 분류할 수 있는 모델을 구현 및 검증하였다. 향후 Bard를 이용한 분류 모델 연구와 제안한 분류 모델을 이용해 프로젝트에 도움이 되는 관리 도구 모델에 관해 연구할 계획이다.

▸**주제어**: 커밋 메시지, GPT, 소스 변경, 디스틸 버트, 다중 레이블 분류

I. Introduction

소스 버전 관리를 위한 SCM(Source Control Management)의 한 종류인 Git은 Linus Torvalds가 리눅스 커널 관리를 위해 개발한 시스템으로 현재는 프로젝트 개발 및 운영 환경에서 개발 및 유지보수 활동에 이용하고 있다. 이러한 Git은 여러 사용자가 수정하는 소스코드의 변경 내용과 이슈를 관리하면서 변경 사항을 저장하는데 커밋은 한 번의 변경 단위를 말한다[1-2]. 커밋은 개발 및 단위테스트 단계에서 신규 기능 및 테스트 코드가 순차적으로 저장되고, 프로젝트 운영 중에는 프로그램 개선이나 버그 수정과 같은 내용이 기록된다. 이로인해 저장된 커밋 이력과 프로젝트는 밀접한 관련이 있다. 다시 말해, 커밋 이력을 분석함으로써 프로젝트와 운영활동간 리스크를 줄이고 프로젝트 현황을 파악하며 비용 절감 및 시간 효율을 높일 수 있는데, 이에 관련하여 다양한 연구들이 진행되고 있다[3]. 본 논문에서는 많은 커밋 분석 연구 중 커밋 메시지를 유지보수 관리 행위의 유형 세 가지로 분류하는 모델을 연구하였고, 커밋 분류 모델을 이용한 솔루션 등의 활용을 목적으로 진행하였다[4]. 솔루션 활용을 위해서는 비용과 밀접하게 관련이 있어 정확도의 신뢰성이 필요하여 이전 연구에서는 정확도를 높이는 목적으로 기존 연구들의 장점들을 조합하여 정확도 높은 분류 모델을 구현하였다. 하지만 해당 모델의 프로세스 중 소스 변경 내용을 추출하는 부분은 JAVA언어 외에는 추출이 안 되기 때문에 결국은 JAVA로 작성된 프로젝트 외에는 분류할 수 없는 문제가 있다. 이를 해결하기 위해 소스 변경 내용을 추출하는 부분을 GPT로 대체함으로써 언어의 제약을 없애고 정확도를 높일 수 있는 모델을 구현하였다. 본 논문의 최종 목적인 깃 연계 프로젝트 지원 도구를 위해서는 프로젝트를 JAVA로 한정할 수 없기 때문에 언어의 제약을 없애는 개선 사항은 시스템 도입에 도움이 될 것이다.

II. Preliminaries

1. Related works

1.1 Software Maintenance Type

다음 표 1 은 소프트웨어 유지관리의 유형 세 가지를 설명한다[4].

Table 1. Software Maintenance Type

Category	Description
Corrective	Fix Bug
Perfective	Performance Improvement
Adaptive	Feature Addition

소프트웨어 유지관리의 활동은 다음과 같다.

첫째, 코렉티브(Corrective)는 문제가 되는 기능의 버그를 교정 및 고치는 유지보수 활동 행위이다.

둘째, 퍼펙티브(Perfective)는 시스템의 보완하기 위해 개선하는 유지보수 활동 행위이다.

셋째, 어댑티브(Adaptive)는 기존 시스템에 신규 기능을 추가하는 개발 유지보수 행위이다.

본 논문에서는 커밋 메시지를 유지보수 활동 세 가지 형태로 분류하는 연구이다.

1.2 Commit Message Classification Model By Utilizing Source Code Change

다음 그림 1 은 소스 변경 데이터를 추출하여 커밋 메시지를 분류하는 모델의 프로세스이다.

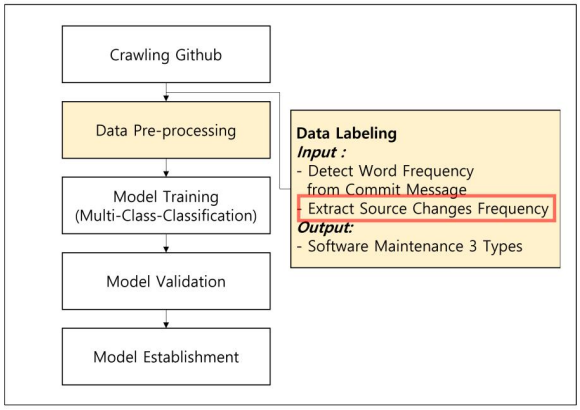

Fig. 1. Classification Process Using Source Change

이 연구에서는 체인지디스틸러(ChangeDistiller)라는 소스 변경 추출 도구를 이용한다. 체인지디스틸러는 JAVA로 작성된 소스 코드의 변경 전 소스와 변경 후 소스를 입력하면 변경된 사항을 48의 크기인 배열로 제공해준다. 다음 표 2 는 체인지디스틸러가 제공하는 배열의 인덱스마다 뜻하는 소스 변경 타입 48가지 유형 중 일부이다[5-8].

Table 2. Source Change Type Index Description

Idx	Change Type	Description
3	ADDITIONAL_CLASS	Class Added
4	ADDITIONAL_FUNCTIONALITY	Function Added
5	ADDITIONAL_OBJECT_STATE	Object State Added
6	ALTERNATIVE_PART_DELETE	Alternative Part Removed
7	ALTERNATIVE_PART_INSERT	Replacement Part Added
9	ATTRIBUTE_TYPE_CHANGE	Attribute Type Modified
10	CLASS_RENAMING	Class Renamed
21	METHOD_RENAMING	Method Renamed
22	PARAMETER_DELETE	Parameter Deleted
23	PARAMETER_INSERT	Parameter Added
25	PARAMETER_RENAMING	Parameter Renamed
33	REMOVED_CLASS	Class Removed
34	REMOVED_FUNCTIONALITY	Function Removed
42	STATEMENT_DELETE	Statement Deleted
41	STATEMENT_INSERT	Statement Added

제공한 배열에는 변경된 유형의 인덱스에 빈도수를 제공한다. 일반적인 커밋 메시지 분류 모델에 소스 변경 빈도 배열을 학습할 때 특징(Feautre)으로 같이 활용하였고 정확도(Accuracy) 76%로 보고하였다[9]. 본 논문에서는 제안하는 논문과 비교하기 위해 해당 모델을 직접 구현하여 같은 커밋 목록들을 대상으로 정확도를 비교한다.

1.3 Commit Message Classification By Utilizing DistilBERT

다음 그림 2 는 전이학습 모델인 구글(Google)의 디스틸 버트(DistilBERT)를 이용한 분류 모델이다.

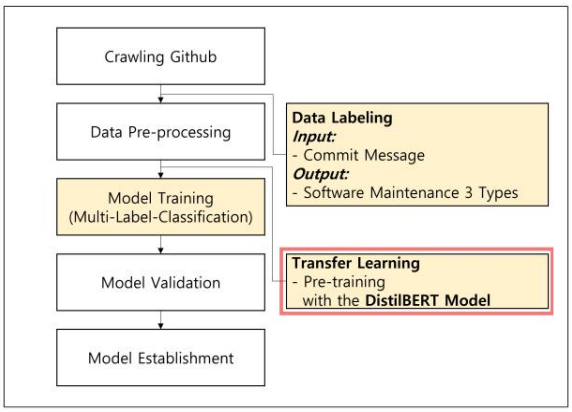

Fig. 2. Classification Process Using DistilBERT Model

이 연구에서는 디스틸 버트 전이학습 모델을 사용해 커밋 메시지를 분류하였다. 디스틸 버트는 버트를 커스터마이징한 버트 모델로 버트보다 빠르고 가벼운 모델이다[10-11]. 많은 데이터를 학습하기에 적합하고 효과적으로 사용될 수 있다. 해당 모델은 입력값에 커밋 메시지만을 입력하여 유지관리 세 가지 유형으로 레이블 하여 지도 학습한 모델이다. 이 연구에서는 정확도 추출을 F1 점수로 평가하였고 평가된 점수는 87%로 보고하였다[12].

1.4 Commit Message Classification That Combines The Two Model.

다음 그림 3 은 1.2의 모델과 1.3 두 가지 모델의 특장점을 조합한 분류 모델의 프로세스이다.

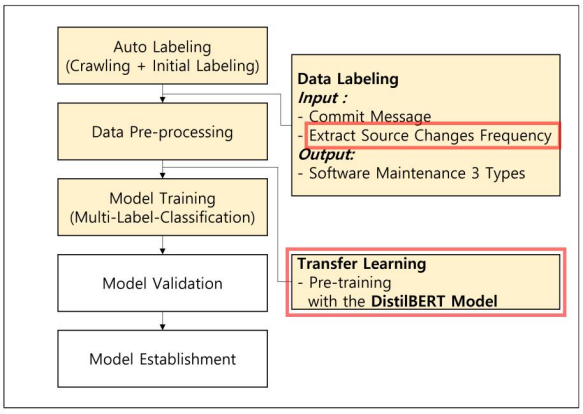

Fig. 3. Proposed Classification Process

이 연구에서는 관련연구 1.2 분류 모델과 1.3 분류모델에서의 장점들을 조합한 모델이다. 1.2 연구에서는 학습 시 체인지디스틸러를 이용해 JAVA 언어를 대상으로 소스 변경 빈도 배열을 특징으로 이용하고 1.3 분류모델에서의 디스틸버트를 이용해 전이학습을 이용하였다. 다음 그림 4는 학습 시 레이블링한 데이터 일부이다.

commitMsg	changeType	Corrective	Perfective	Adaptive
add file delete retry to testcluster elasticsearchnode (#89095) add retry logic for cleanup / deletion in testcluster's elasticsearchnode, to tolerate the asynchronous nature of deletions on the windows file-system.	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	0	0	1
fix object equals for sqlqueryrequest's binarycommunication (#87887) co-authored-by: owenniceliu <owenniceliu@tencent.com>	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]	1	0	0
...	...	...	...	...

Fig. 4. Labeled Data

이 연구에서는 정확도를 F1점수로 평가하였고 95%로 보고하였다[13]. 본 논문에서는 해당 연구에서 JAVA 언어로 작성된 소스를 대상으로만 모델을 적용할 수 있는 문제를 개선하기 위해 GPT를 이용해 소스 변경 요약내용을 추출하여 언어제약을 없앨 수 있다.

III. The Proposed Scheme

1. System Configuration

다음 그림 5 는 본 논문에서 구성한 GPT를 이용한 분류 모델의 시스템 구성도이다.

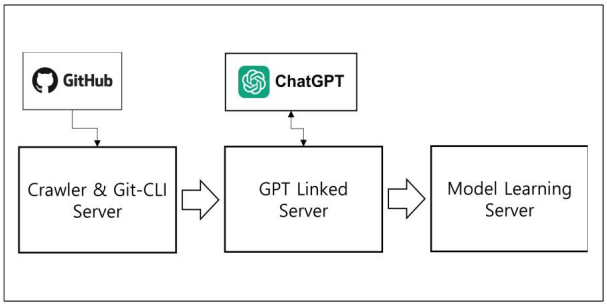

Fig. 5. System Configuration

크롤러(Crawler) 서버는 Github에서 선정된 프로젝트를 클론(Clone)하고 프로젝트마다 커밋 정보를 추출하고, 추출한 커밋 정보는 커밋 ID, 커밋 메시지, 실제 소스의 변경 소스이다. GPT 연계 서버는 ChatGPT 서비스를 API로 연계한 서버이다. 해당 서버는 Spring Boot로 구성된 톰캣 WAS(Web Application Server) 이고 실제 소스 내용을 입력받아 ChatGPT 서비스에 API 통신을 하여 소스의 변경된 요약 정보를 응답받아 정보를 취합한다. 모델 학습 서버는 파이썬 서버로 구성했고 실질적인 모델 학습을 담당한다. 모델 학습 서버는 앞의 두 개의 서버를 거쳐 추출하고 취합하여 가공한 데이터를 이용하여 학습을 진행한다. Python으로 작성된 학습 서버는 AI 및 데이터 분석에 많이 활용되는 라이브러리르 이용해 개발하였다. numpy와 Pandas, simpletransformers 라이브러리를 이용해 데이터 전처리 및 학습과 검증을 수행하고, tensorboard, Seaborn등의 라이브러리를 이용해 학습 과정과 학습 결과를 선형차트로 표현했다.

2. Target Languages

분석 대상의 언어는 티오베(TIOBE) 지수에서 2023년 1월부터 9월까지 프로그래밍 언어 순위 1위를 유지하고 있는 파이썬과 지난 연구와의 비교를 위해 JAVA언어를 선정하였다. 다음 표 3 은 티오베 지수의 언어 별 월의 지수를 나타낸다[14].

Table 3. 2023 Year Language Usage Rate

Language	Jan	May	Aug	Sep
Python	16.3%	13.5%	13.3%	14.1%
C	16.2%	13.4%	11.4%	11.2%
JAVA	12.2%	12.2%	10.3%	9.5%
C++	12.9%	12.0%	10.6%	10.7%
C#	5.7%	7.4%	7.0%	7.3%

3. Target Projects

분석 대상 프로젝트는 다음 표 4 과 같다.

Table 4. Target Projects

No	Language	Project Name	Star	Fork
1	Java	Elasticsearch	61.4k	22k
2	Java	Graal	17.6k	1.4k
3	Java	Platform Frameworks Base	10.1k	6k
4	Python	DeepFaceLab	41.5k	9k
5	Python	AirFlow	31k	12k

본 논문에서 프로젝트는 관련 연구 1.4 분류 모델에서 선택했던 JAVA 기반의 세 가지 프로젝트와 현재 프로그래밍 인기 순위가 가장 높은 파이썬으로 작성한 두 가지 프로젝트를 추가하였다. 다섯 개의 프로젝트는 Github 에서 공개된 오픈소스이다. 별점(Star)과 인용 수(Fork)는 프로젝트의 추천한 수와 프로젝트를 인용하고 커스터마이징한 점수를 나타내는데 프로젝트의 신뢰 높은 프로젝트임을 알 수 있는 지표이다. 본 논문에서 선정한 다섯 개의 프로젝트는 모두 10000명 이상의 개발자가 추천한 프로젝트이고 해당 프로젝트를 인용하는 개수도 1000개 이상부터 많게는 20000개의 인용한 프로젝트가 있다. 선정된 다섯 개의 프로젝트에서는 커밋 데이터 학습을 위해 프로젝트마다 500건씩 총 2500건의 데이터를 추출하였다.

4. System Process

다음 그림 6 은 구현 모델의 시스템 프로세스이다.

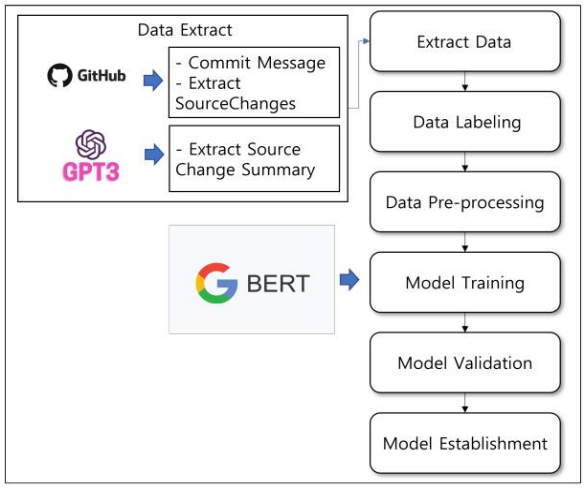

Fig. 6. Proposed Classification Process

전체적인 프로세스는 데이터 추출, 데이터 레이블링, 데이터 전처리, 모델 학습, 모델 검증, 모델 수립의 순서로 진행된다. 단계에 대한 상세한 내용은 다음과 같다.

4.1 Extract Data

4.1.1 Commit Basic Info Extraction

다음 그림 7 은 커밋 정보를 추출하기 위한 명령어이다.

```
git log --name-only\
--pretty="format:####START####%h%n%B%n####END####" \
--until="2021-07-21" \
-500 -- '*.java' \
> githistory_java_elastic_withBody.out
```

Fig. 7. Git Log Command

Git CLI(CommandLine Interface)는 많은 기능을 제공하는데 그 중 커밋의 정보를 추출하기 위해 git log 명령어를 사용하여 일괄 추출할 수 있다[15-16]. log 명령어는 옵션을 통해 원하는 형태로 출력이 가능한데 –name-only 명령어와 pretty 옵션을 통해 원하는 파싱 형태로 가공 했다. name-only 옵션은 변경된 소스 파일의 목록을 제공하고 pretty 옵션은 출력 포맷을 지정한다. 다음 그림 8 은 추출한 깃의 커밋 정보 세트의 예제이다.

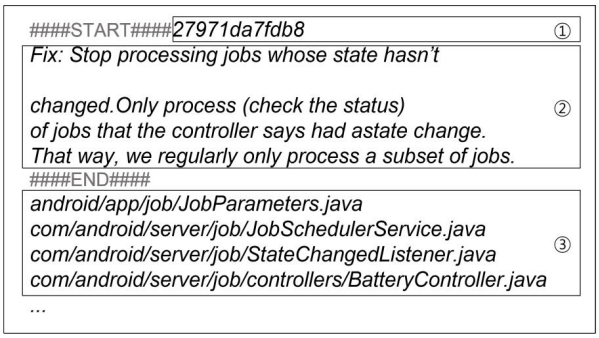

Fig. 8. Commit Info Set Example

(1)번은 커밋 ID이고, (2)번은 커밋을 수행한 작성자가 남긴 커밋 메시지이고 (3)번은 커밋에서 수정된 소스 파일의 목록을 보여준다. 커밋 하나마다 그림 8과 같이 하나의 세트로 정보를 출력한다. 본 논문에서는 이와 같은 형태로 프로젝트마다 500개의 커밋 정보 세트를 추출했다.

4.1.2 Source Change Extraction

다음 그림 9 는 소스 변경 내용을 추출하는 명령어이다.

```
git diff \
27971da7fdb8 android/app/job/JobParameters.java
```

Fig. 9. Command to extract source change

diff 명령어는 2개의 인자를 받는데 첫 번째 인자로 커밋의 ID를, 두 번째 인자로 해당 커밋에 포함된 소스파일의 경로와 파일명을 입력한다. 추출된 소소의 샘플은 다음 그림 10 와 같다.

```
@@ -68,9 +85,16 @@
public final class BatteryController
            extends RestrictingController {
        …
        mTrackedTasks.add(taskStatus);
        taskStatus.setTrackingController(
                JobStatus.TRACKING_BATTERY);
-       taskStatus.setChargingConstraintSatisfied(nowElapsed,          ①
-               mChargeTracker.isOnStablePower());
-       taskStatus.setBatteryNotLowConstraintSatisfied(
-               nowElapsed, mChargeTracker.isBatteryNotLow());
+       taskStatus.setBatteryNotLowConstraintSatisfied(nowElapsed,     ②
+       mService.isBatteryNotLow());
    }
  }
```

Fig. 10. Source Change Extraction

(1)번 영역은 해당 파일에서 삭제된 부분을 표현한다. 마이너스 기호를 맨 앞에 붙이고 해당 소스내용을 보여준다. (2)번 영역은 추가된 소스 내용을 표현하는데 삭제 표현과 반대로 플러스 기호를 통해 소스의 추가를 표현한다.

4.2 Labeling

다음 표 5 는 수집한 데이터와 레이블링한 데이터를 정리한 CSV(Comma-Seperated Values)파일의 의 컬럼 정보이다.

Table 5. Target Projects

No	Column	Description
1	project	Project Name
2	id	Commit ID
3	msg	Commit Message
4	t1	Corrective Type
5	t2	Perfective Type
6	t3	Addaptive Type
7	c1~c11	Change File Info 1~12

1~3번의 컬럼은 추출한 커밋의 프로젝트명과 커밋 ID 그리고 커밋 메시지이다. 4~6의 컬럼은 분류하고자 하는 커밋 분류 유형 세 가지로 Corrective, Perfective, Adaptive를 1,0의 Boolean 값으로 표현하였다. 분류 유형은 관련 연구 1.1 Software Maintenance Type에서 자세히 설명하였다. 7번 컬럼은 실제로 11개의 컬럼으로 구성하였으며 각 컬럼에는 해당하는 커밋에서 실제 수정된 파일의 변경 내역이 로그로 저장되어 있다. 11개로 구성한 이유는 각 컬럼은 커밋에서 변경한 파일의 수만큼 구성할 수 있는데 수집한 커밋 목록에서의 가장 많은 파일 수를 가진 커밋이 11개이기 때문에 11개의 컬럼을 구성하여 커밋 당 모든 파일의 변경된 내용을 저장하였다. 총 수집된 데이터 개수는 5개의 프로젝트에서 각각 300개의 커밋을 수집하여 총 1500건의 데이터를 수집하였고 해당 과정에서 업무 담당자들이 직접 커밋을 분류 및 검증을 하고 학습에 불필요한 커밋 정보는 삭제하였다. 학습에 불필요한 데이터 중 대표적인 예는 브랜치를 병합할 때 자동으로 생기는 커밋 메시지인 “Merge” 키워드로 시작하는 커밋 메시지가 있고, 커밋 내용 중 변경된 파일이 없는 것을 제거하였다. 해당 과정에서 남은 최종 데이터는 827건이다.

4.3 GPT-Linked Extraction

다음 그림 11 은 GPT를 연계한 소스 코드의 일부 내용이다.

```
chunks = \                                                    ①
  split_source_code(source_code, max_tokens=4096)
conversation = \
  [{"role": "system", "content": default_question}]
for chunk in chunks:
    conversation\
       .append({"role": "user", "content": chunk})

try:                                                          ②
  response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=conversation
  )
  result = \
    response.choices[0].message["content"]
except Exception as ex:
  result = str(ex)
```

Fig. 11. GPT-Linked Code

(1)번 영역은 GPT 3.5의 경우 총 토큰 수는 4096개로 제한되기 때문에 4096개 보다 토큰 수가 많은 경우 메시지를 4096개 토큰으로 나누어 모든 소스 코드를 분석할 수 있게 한다. 그리고 소스 코드를 입력받은 GPT가 소스 코드의 변경 내용을 요약한 결과를 응답받기 위해 소스 코드를 입력하기 전 Default_Question값으로 “Here is the source change using Git's diff command. Please explain the purpose of the content.”의 질문을 먼저 요청한다. (2)번 영역은 GPT API와 연동해서 소스 변경의 요약내용을 응답받을 수 있는 함수를 호출하는 부분이다. 사용하는 GPT 모델은 gpt-3.5-turbo를 사용하였고 openai 라이브러리를 활용해 GPT API를 호출하였다.

4.4 Data Pre-Processing

다음 그림 12 는 전처리 과정의 소스 코드이다.

```
train_df['t1'] = train_df['t1'].apply(lambda x: int(x))                    ①
train_df['t2'] = train_df['t2'].apply(lambda x: int(x))
train_df['t3'] = train_df['t3'].apply(lambda x: int(x))
train_df = train_df.apply(lambda x : x.fillna(''))

train_df['labels']= \                                                      ②
  train_df[['t1','t2','t3']]  \
  .values.tolist()

train_df['text'] = \                                                       ③
  "[CLS] " + train_df['msg'] \
    + " [SEP] \n\n" + train_df['c1_rs'] \
    + " [SEP] \n\n" + train_df['c2_rs'] \
    + " [SEP] \n\n" + train_df['c3_rs'] \
    + " [SEP] \n\n" + train_df['c4_rs'] \
    + " [SEP] \n\n" + train_df['c5_rs'] \
    + " [SEP] \n\n" + train_df['c6_rs'] \
    + " [SEP] \n\n" + train_df['c7_rs'] \
    + " [SEP] \n\n" + train_df['c8_rs'] \
    + " [SEP] \n\n" + train_df['c9_rs'] \
    + " [SEP] \n\n" + train_df['c10_rs'] \
    + " [SEP] \n\n" + train_df['c11_rs'] \
    + " [SEP]"

train_df = train_df[['text','labels']]                                     ④
train, test = train_test_split(train_df, test_size=0.20)
```

Fig. 12. Pre-Processing Source Code

디스틸버트 모델은 대량의 단어 임베딩에 대해 사전 학습이 되어 있는 모델을 제공한다. 따라서 데이터 대부분의 전처리 과정을 생략할 수 있는 장점이 있다. (1)번 과정에서 학습이 가능하도록 형 변환과 공백 및 Null인 데이터를 모두 제거한다. (2)번 과정에서 분류하고자 하는 세 가지 소프트웨어 유형 세 가지를 “labels” 라고 명명한 컬럼에 배열 형태로 입력한다. (3)번 과정에서 커밋 메시지와 커밋별 GPT로 추출한 11가지의 소스 변경 내용을 버트의 입력값 프로토콜에 맞게 데이터를 가공한다. 해당 과정에서는 입력 데이터가 12가지의 형태이므로 커밋 메시지와 11가지의 소스 변경 내용을 ‘[SEP]’ 이라는 구분자로 구분하여 디스틸 버트모델의 프로토콜에 맞게 변환한다[17-18]. (4)번 과정에서는 과적합(Overfitting)을 막기 위해 8:2 비율로 학습 데이터와 검증데이터로 먼저 분리하여 학습했던 데이터가 검증에 사용하지 않도록 한다.

4.5 Transfer Learning

모델 학습은 하이퍼 파라미터(Hyper Parameter)의 설정에 따라 결과가 다른데, 많은 반복적인 연구를 통해 본 연구 모델에 맞는 최적의 파라미터값을 도출하였다. 도출된 파라미터는 다음 표 6 와 같다.

Table 6. Hyper Parameter Config Values

Config	Value
num_train_epochs	30
learning_rate	2e-05
train_batch_size	8

학습 시 에포크(epoch)수는 30회 설정을 넘어가면서 학습 정확도가 떨어짐을 확인할 수 있다. 학습률(learning rate)은 학습을 반복 시 파라미터 조정 빈도를 뜻하는데, BERT[14] 논문에서 BERT 모델을 이용할 때 학습률이 최적인 설정값을 5e-5, 4e-5, 3e-5, 2e-5라고 보고하고 있다. 본 논문에서는 4개의 설정값을 적용해본 후 가장 좋은 결과를 도출하는 2e-5 값을 설정한다. 배치 크기(batch size)의 경우는 BERT 논문에서 32값이 최적이라고 보고되었지만 본 논문의 연구에서는 8값까지 낮췄을 때 최적의 학습곡선을 그리는 결과를 확인하고 8값으로 설정한다. 학습은 총 827건의 데이터의 80%인 662개의 데이터로 학습하고 배치(Batch) 크기인 8로 나눈 1회의 에포크(Epoch)마다 83개의 스텝(Step)을 30회 진행하여 총2,490회의 반복(Iteration)을 통해 파라미터를 계속 변경한다. 다음 그림 13은 학습은 진행하는 모습이다.

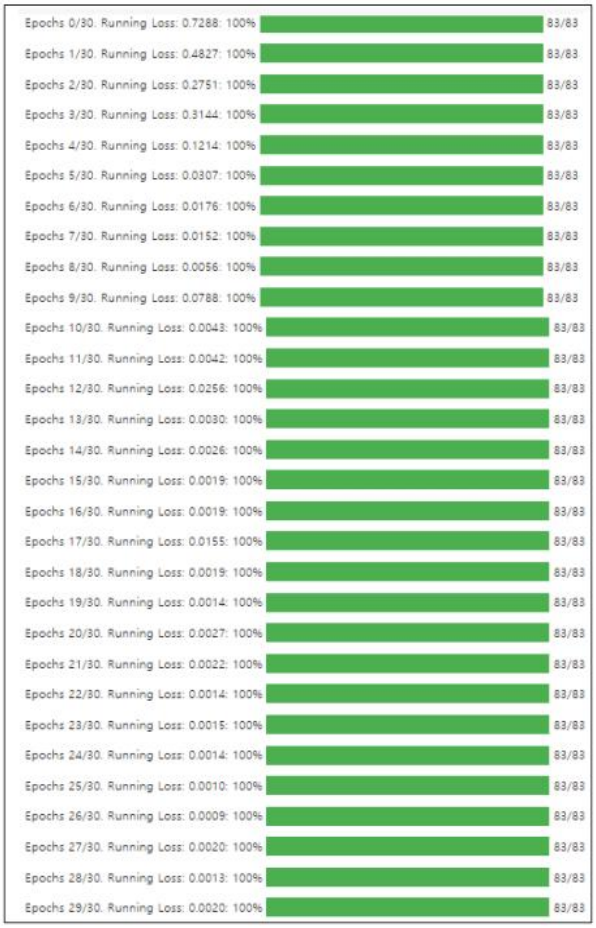

Fig. 13. Learning Progress

다음 그림 14 와 그림 15 는 학습 시 손실률과 학습률 그래프이다.

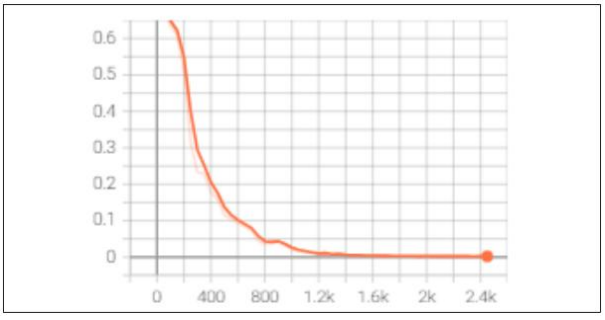

Fig. 14. Loss Rate

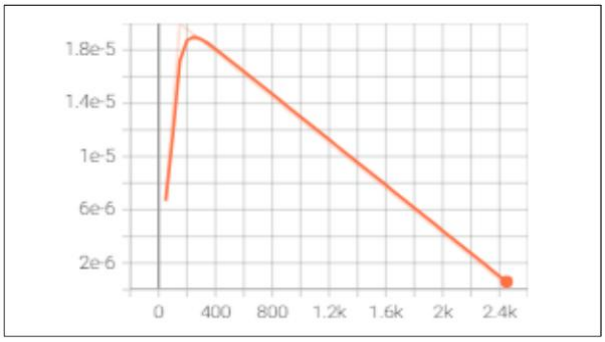

Fig. 15. Learning Rate

30회의 에포크 동안 학습이 계속될수록 손실률은 떨어져서 최종 학습 시에는 0.0020까지 떨어짐을 확인할 수 있다. 학습률은 2e-5까지 Y축으로 선형적으로 증가하며 학습하고 이후에는 0에 수렴하는 모습으로 학습이 정상적으로 이루어짐을 알 수 있다.

IV. Validation

1. Model Validation

해당 과정에서는 학습 데이터로 사용하지 않은 미리 분리 해뒀던 20%의 검증데이터로 모델검증을 진행한다. 다중 레이블 분류 검증에 많이 이용하는 F1 점수를 계산하기 위해 정밀도(Percision)과 재현율(Recall)을 먼저 계산하고 F1 점수를 계산하였다. 다음 표 7 는 제안 모델의 검증 결과이다.

Table 7. Model Validation Results

Item	Score
Percision	91%
Recall.	92%
F1-Score	91.2%

검증 결과의 Percision은 정밀도를 나타내는데 91%의 정밀도 결과를 얻었다. Recall은 재현율로 92%를 기록했다. F1-Score는 F1 점수이다. 정밀도와 재현율의 평균으로 계산된다. F1 점수가 높은 분류 모델이 좋다고 평가할 수 있으며 제안 모델의 검증 결과는 F1 점수인 91%의 결과를 보여준다.

2. Model Evaluation

다음 표 8 은 모델 검증 결과와 다른 모델과 제안 모델의 정확도를 비교한 표이다.

Table 8. Compare Model Results

Dataset	F1-Score	H-Loss
Proposal Model	**91%**	**0.06**
J. H. Choi et al	95%	0.04
M. U. Sarwar et al.	87%	0.11
S. Levin et al.	76%	-
Gharbi et al.	71%	0.22
Mauczka et al.	50%	0.31

이전에 연구했던 JAVA 언어 대상으로만 가능한 J. H. Choi의 디스틸버트와 소스 코드 변경 내용을 함께 학습한 모델보다는 4% 낮은 점수를 기록했다. 비록 F1 점수가 떨어지긴 했지만 제안한 모델은 JAVA가 아닌 다른 언어로 작성된 프로젝트도 제약 없이 분류할 수 있음을 보였다. 그리고 앞서 커밋 메시지를 분류했던 연구들과 비교해보면 M. U. Sarwar의 연구인 디스틸버트만 이용했던 모델보다 4% 정확도 향상이 있으며, 소스 변경 유형 빈도를 이용한 S. Levin의 모델보다 15% 정확도 향상을 나타냈다. 마지막으로, Mauczka의 연구와의 비교에서는 41% 향상된 결과를 보여준다[19-20]. 이는 제안하는 모델이 다양한 비교 연구들과 비교하여 대부분 높은 정확도 향상을 확인할 수 있으며, 다른 언어로 작성된 프로젝트에 대한 분류에서도 그 타당성을 입증할 수 있다.

V. Conclusions

본 논문은 커밋 메시지를 더 정확히 분류하기 위해서 소스 변경 데이터를 같이 활용하였다. 이전 연구에서는 커밋 메시지 분류 모델에 JAVA 언어로 작성된 프로그램만 소스 변경 데이터를 추출할 수 있는 문제가 있었다. 본 연구에서는 이전 연구의 단점인 언어제약을 없애기 위해 생성형 AI인 GPT 모델로 소스 변경 데이터를 자연어로 표준화시키는 전처리 단계를 추가하여 언어제약을 없앴고 모델의 검증을 위해 JAVA로 작성된 프로그램 외에도 Python으로 작성된 프로그램들을 추가 대상으로 데이터를 수집하여 모델을 학습하였고 모델의 점수를 91%까지 도출하였다. 다른 연구들과 비교 시 이전 연구에서 획득했던 모델점수보다 4% 낮은 평가를 기록하였지만 특정 프로그램 언어에 종속되지 않고 분류할 수 있는 모델을 제안 및 검증하였다. 하지만 비교한 논문의 경우 분류 성능이 크게 다를 수 있고, 각 모델은 서로 다른 맥락에서의 분석인 부분과 다른 데이터 세트로 테스트한 차이점이 있고, 모델 구현에 대한 정확한 재현이 어렵기 때문에 실증적 분석이 어렵다. 이로 인해 도출한 비교 결과가 편향된 결과일 가능성이 있다[21]. 추후 구현이 가능한 모델들을 최대한 유사하게 구현하여 동일 데이터 샘플로 학습시킨 실증적 비교가 필요하다. 또한, GPT 모델과 유사한 생성형 AI 모델인 구글의 바드(Bard) 모델을 이용해서 정확도의 차이를 연구하고 더 나아가 제안하는 분류 모델을 이용해 프로젝트에 도움이 되는 관리 도구 모델에 관해 연구할 예정이다.

REFERENCES

[1] Heričko, T., & Šumak, B, "Commit classification into software maintenance activities: A systematic literature review," In 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC) IEEE, pp. 1646-1651, 2023, June. DOI: 10.1109/COMPSAC57700.2023.00254

[2] T. Heričko, "Automatic data-driven software change identification via code representation learning," Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, 2023. DOI: 10.1145/3593434.3593505

[3] Mockus and Votta, "Identifying reasons for software changes using historic databases," Proceedings 2000 International Conference on Software Maintenance, pp. 120-130, 2000. doi: 10.1109/ICSM.2000.883028.

[4] S. Gharbi, M. W. Mkaouer, I. Jenhani, and M. B. Messaoud, "On the Classification of Software Change Messages Using Multi-Label Active Learning," in Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 1760–1767, 2019. doi: 10.1145/3297280.3297452

[5] H. C. Gall, B. Fluri, and M. Pinzger, "hange analysis with evolizer and changedistiller," IEEE Software, vol. 26, no. 1, p. 26, 2009. DOI: 10.1109/MS.2009.6

[6] B. Fluri, E. Giger, and H. C. Gall, "Discovering patterns of change types," in Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on. IEEE, pp. 463–466, 2008. DOI: 10.1109/ASE.2008.74

[7] M. Martinez, L. Duchien, and M. Monperrus, "Automatically extracting instances of code change patterns with ast analysis," arXiv preprint arXiv:1309.3730, 2013. DOI: 10.1109/ICSM.2013.54

[8] B. Fluri, M. Wursch, M. PInzger, and H. C. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," Software Engineering, IEEE Transactions on, vol. 33,

no. 11, pp. 725-743, 2007. DOI: 10.1109/TSE.2007.70731

[9] S. Levin, and A. Yehudai, "Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes," PROMISE: Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, pp. 97-106, November. 2017. doi: 10.1145/3127005.3127016

[10] A. Adhikari, A. Ram, R. Tang, and J. Lin, "DocBERT: BERT for Document Classification," arXiv, 2019. doi: 10.48550/ARXIV.1904.08398

[11] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," arXiv preprint arXiv:1910.01108, 2019. DOI: 10.48550/arXiv.1910.01108

[12] M. U. Sarwar, S. Zafar, M. W. Mkaouer, G. S. Walia and M. Z. Malik, "Multi-label Classification of Commit Messages using Transfer Learning," 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 37-42, 2020, doi: 10.1109/ISSREW51248.2020.00034.

[13] J. H. Choi, J. Y. Kim and S. H. Park, "Implementation of Git's Commit Message Complex Classification Model for Software Maintenance," Journal of the Korea Society of Computer and Information," Vol. 27, no. 11, pp. 131-138, 2022. doi:10.9708/JKSCI.2022.27.11.131.

[14] Tiobe, https://www.tiobe.com/tiboe-index, 2023.

[15] Hultstrand, S., & Olofsson, R. "Git-CLI or GUI: Which is most widely used and why?," 2015

[16] Miller, C. G. "Introduction to Git". 2022.

[17] Devlin, Jacob, et al. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1, pp. 4171-4186, June. 2019. DOI:10.18653/v1/N19-1423

[18] Sun, Chi, et al. "How to Fine-Tune BERT for Text Classification?," Lecture Notes in Computer Science, pp. 194-206. June. 2019. DOI:10.1007/978-3-030-32381-3_16

[19] A. Mauczka, F. Brosch, C. Schanes, and T. Grechenig, "Dataset of developer-labeled commit messages," in 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, pp. 490-493, 2015. DOI: 10.1109/MSR.2015.71

[20] S. Zafar, M. Z. Malik, and G. S. Walia, "Towards standardizing and improving classification of bug-fix commits," in 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, pp. 1-6, 2019. DOI: 10.1109/ESEM.2019.8870174

[21] Li, J., & Ahmed, I, "Commit message matters: Investigating impact and evolution of commit message quality," In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) IEEE, pp. 806-817, 2023, May. DOI: 10.1109/ICSE48619.2023.00076

Authors

Ji-Hoon Choi received his master's degree in computer Engineering from Kongju University in March 2020. Currently, the doctoral program in Computer Science and Engineering at Kongju University is in progress from March 2021. Ji-Hoon Choi started programming as a Web-based JAVA program in 2014. Currently, as a manager at Ice Cream Kids Company, he is in charge of developing and operating B2C services and LCMS related to Early-childhood-education. He is interested in project management systems, automated testing, algorithms, machine learning, and automated systems.

Jae-Woong Kim received the bachelor's degree and the M.S. degree in the Department of Computer Engineering from the Jungang University in 1983 and 1988, respectively. He received the Ph.D. degree in the Department of Computer Engineering from Daejun University in 2000. He has been a professor in the Department of Computer Engineering at Kongju National University since 1992. His current research interests include software engineering.

Seong-Hyun Park received the B. S. degree in College of Arts and Music from Chungnam National University, Korea in 2017. The M. S, and Ph. D. degrees in Computer Engineering from Kongju National University, Korea, in 2017, 2020. He is currently a teaching in the Department of Computer Science & Engineering. Kongju National University. He is interested in computer music, convergence Education. and Real Time System and Management and Clout computing and Communication.