LLM을 활용한 소프트웨어 문서 분류 및 생성 연구

경상국립대학교 ▌허주은·이선아*

1. 서 론

맥킨지 글로벌연구소에서 2023년 6월에 발표한 'The Economic Potential of Generative AI[1]'에 따르면, 생성형 AI는 연간 2조 6천억 달러에서 최대 4조 4천억 달러의 가치를 창출할 수 있다고 예측했다. 또한, 업무 자동화를 통해 작업자의 업무 시간을 60~70% 단축할 수 있는 잠재력을 지니고 있으며, 이를 통해 개인 작업자의 역량을 강화하고 업무의 구조를 바꿀 수 있을 것으로 예상했다. 보고서에서는 특히 고객 관리, 영업, 소프트웨어 엔지니어링, 연구 개발 분야에서 생성형 AI의 활용이 빛을 발할 것으로 전망했다. 소프트웨어 공학에서 생성형 AI를 활용하면 반복적인 작업을 자동화 하고, 복잡한 작업을 효율적으로 처리할 수 있다. 이러한 거대 언어 모델의 발전에 따라 소프트웨어 공학 도메인의 다양한 태스크에 대한 자동화 시도가 일어나고 있다.

소프트웨어 공학 원칙과 기술은 시스템의 복잡성을 관리하고 품질보증, 효율적인 개발 프로세스 및 유지보수 확장성을 가능하게 한다. 소프트웨어 공학의 고질적인 문제는 소프트웨어의 비가시성과 복잡성으로 인해, 소프트웨어 개발 주기 동안 많은 양의 개발 산출물을 작성해야 한다는 점이다. 또한, 시스템의 변경이 발생할 때 이러한 산출물들을 일관성 있게 변경해야 하는 변경관리의 수고로움이 있다. 이러한 프로젝트 관리자와 개발자의 부담을 덜기 위해 소프트웨어 개발 방법론은 과거 문서 중심의 전통적인 방법에서 사용자와의 소통을 중심으로 하는 애자일 방법으로 개선되고 있지만, 여전히 프로젝트 관리자와 개발자가 작성해야 할 문서의 양은 많다. 정해진 소프트웨어 개발 시간 동안 요구되는 산출물을 모두 작성하는 것은 관리자와 개발자에게 큰 노력과 시간을 요구한다.

이러한 문제를 해결하기 위해 소프트웨어 공학 도메인에서도 다양한 기술을 활용한 자동화에 관한 연구가 활발히 진행 중이다. 앞의 매킨지 글로벌연구소에서의 예측과 같이 소프트웨어 산출물에 대한 다양한 분류 및 생성 연구가 진행되고 있다. 그 대상이 되는 소프트웨어 산출물은 표 1과 같다. 초기에는 소프트웨어 산출물을 자동으로 분류하고 생성하기 위한 기계학습/딥러닝 기법을 적용하는 시도가 있었다. 최근에는 대규모 데이터로 학습된 대규모 언어 모델을 사용하는 연구들이 많이 소개되고 있으며, 그 효용성과 효

표 1 소프트웨어 개발 단계에 따라 요구되는 산출물 목록

분석	사용자 요구사항 정의서
	유스케이스 명세서
	요구사항 추적표
설계	클래스 설계서
	사용자 인터페이스 설계서
	컴포넌트 설계서
	인터페이스 설계서
	아키텍처 설계서
	총괄시험 설계서
	시스템 시험 시나리오
	엔티티 관계 모형 기술서
	데이터베이스 설계서
	통합시험 시나리오
	단위시험 케이스
	데이터 전환 및 초기 데이터 설계서
구현	프로그램 코드
	단위 시험 결과서
	데이터베이스 테이블
시험	통합시험 결과서
	시스템 시험 결과서
	사용자 지침서
	운영자 지침서
	시스템 설치 결과서
	인수시험 시나리오
	인수시험 결과서

* 종신회원

† 본 과제(결과물)는 교육부와 한국연구재단의 재원으로 지원을 받아 수행된 3단계 산학연 협력 선도대학 육성사업(LINC 3.0)의 연구결과입니다.

1) https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier#business-value

율성을 입증하고 있다.

우리는 본 원고를 통해 대규모 사전 훈련 모델에 대한 기본 개념을 소개하고, 소프트웨어 개발 주기의 산출물 분류와 생성을 위한 자동화에 관한 최신 연구 문헌의 발전 동향을 탐구한다. 이러한 연구 동향 파악을 통해 소프트웨어 공학 분야에서 대규모 언어 모델 적용과 향후 발전 방향에 대한 이해를 높인다.

원고 구성은 다음과 같다. 2절에서는 문헌에서 사용된 대규모 사전 훈련 모델에 관해 간략히 소개한다. 3절에서는 대규모 언어 모델을 활용하여 자동화 가능한 소프트웨어 공학 태스크를 설명한다. 4절에서는 대규모 언어 모델을 활용한 소프트웨어 공학 문서 분류 및 생성에 관한 최근 연구를 요약하고 소개한다. 5절에서는 4절의 연구 추세에 기반하여 향후 필요한 연구 방향에 대해 논의한다. 마지막으로, 6절에서는 본 원고를 요약하며 마무리한다.

2. 대규모 사전 훈련 모델

본 절에서는 최신 연구에서 많이 활용하고 있는 대규모 언어 모델(LLM)의 간략한 개념을 소개한다. 문헌에서는 주로 트랜스포머(Transformer)[1] 기반 자연어 처리 모델을 기반으로 한 BERT와 GPT 모델을 사용한다. 최근 대규모 언어 모델을 위해서는 프롬프트 엔지니어링과 검색 증강 생성 등의 기법이 사용된다.

2.1 트랜스포머 (Transformer)

트랜스포머는 어텐션 메커니즘을 적용하여 주어진 단어와 다른 단어 간의 관계를 고려한 임베딩을 생성할 수 있고, 입력 시퀀스를 병렬로 동시에 처리할 수 있다. 트랜스포머는 인코더와 디코더 블록으로 구성되어 있으며, 인코더에서는 입력 시퀀스를 받아들이고 이를 의미 벡터로 변환하여 디코더에 전달하는 역할을 디코더는 인코더에서 생성된 의미 벡터를 받아 이를 사용하여 출력 시퀀스를 생성하는 역할을 수행한다. 트랜스포머의 파생 모델인 BERT와 GPT는 2.2 및 2.3절과 같다.

2.2 BERT (Bidirectional Encoder Representations from Transformer)

BERT[2]는 트랜스포머의 인코더 블록으로 구성되어 있으며 Book Corpus(8억 단어)와 English Wikipedia (25억 단어)로 사전 훈련하였다. BERT는 문장을 이해하는 것에 강점을 보이며 특히 문장 분류, 이름 인식, 질의응답, 텍스트 유사도, 단어 의미 추론 등 분석과 이해에 초점을 맞춘 태스크에서 강력한 성능을 보인다.

2.3 GPT-3 (Generative Pretrained Transformer)

GPT-3[3]는 트랜스포머의 디코더 블록으로 구성되어 있으며 책과 인터넷의 데이터 세트에서 수집한 570GB 텍스트 데이터를 사용하여 사전 훈련되었으며, 이는 약 300B의 토큰에 해당한다. GPT는 문장을 생성하는 것에 강점을 보이며 특히 텍스트 생성, 텍스트 요약, 기계 번역, 대화형 AI 등 생성과 생성형 대화에 강력한 성능을 보인다.

2.4 프롬프트 엔지니어링 (Prompt Engineering)

ChatGPT를 특정 작업에 맞게 조절하는 방법이다. 개선된 프롬프트를 사용하면 ChatGPT가 원하는 작업을 더 명확하게 수행할 수 있다. 프롬프트 엔지니어링을 통해 작업을 정의하고 모델의 행동을 조절함으로써 모델이 사용자의 요구에 부합하도록 할 수 있다.

2.5 검색 증강 생성 (RAG, Retrieval-Augmented Generation)

RAG는 정보 검색과 생성 기능을 결합한 접근 방식이다. RAG는 검색 시스템과 생성 모델을 통합하여 더 정확하고 풍부한 정보를 제공한다. RAG의 주요 구성 요소는 먼저 검색 시스템 (Retrieval System)으로 사용자의 질의에 대해 관련된 문서나 정보를 검색한다. 다음으로 생성 모델 (Generation Model)은 검색된 정보를 바탕으로 자연어 답변을 생성한다.

3. LLM의 소프트웨어 공학 태스크 적용 예시(LLM4SE)

현재 소프트웨어 공학 도메인에서 대규모 사전 훈련 모델을 접목하여 소프트웨어 공학의 복잡하고 노동집약적인 태스크를 자동화하여 효율성과 정확도를 높이기 위한 시도들이 수행되고 있다. ‘Generative AI in Software Industry-Part 1’ 기사[2]에 의하면 AI 기술을 적용할 수 있는 소프트웨어 공학 태스크에 따른 세부 태스크를 명시한다. 해당 기사를 참고하여 각 세부 태스크에 따른 소프트웨어 개발 단계와 AI의 역할을 표 2와 같이 정리할 수 있다.

4. 소프트웨어 공학 태스크 자동화 연구

대규모 언어 모델을 활용하여 소프트웨어 공학 태스크를 자동화하는 연구를 본 연구실에서

2) https://medium.com/@manish.shanker81/generative-ai-in-software-industry-part-1-3db26ac435ae

표 2 AI 기술을 활용한 소프트웨어 공학 태스크 자동화

SE 단계	태스크	세부 태스크	AI의 역할
설계	소프트웨어 설계	소프트웨어 아키텍처 자동생성	디자인 요구사항을 분석하여 소프트웨어 아키텍처를 자동 생성함
		지능형 코드 리팩토링 제안	코드 구조를 분석하고 리팩토링 제안을 제공함
		코드 스멜 탐지 및 개선	코드 스멜을 탐지하고 이를 개선함
		디자인 패턴 추천	시스템 특징에 따라 적절하고 최적화된 디자인 패턴을 추천함
		아키텍처 자동 합성	소프트웨어 아키텍처를 자동으로 생성하고 이들을 연동함
		AI 기반 의사결정 지원	소프트웨어 설계 의사결정을 다양한 사례를 통해 도움
개발	코드 생성	API 자동 생성	데이터 모델, 디자인 패턴, 소스코드 기반으로 API 엔드포인트를 자동 생성함
		코드 스니펫 자동 생성	코드 패턴을 학습하여 필요한 코드 스니펫을 자동 생성함
		단위 테스트 자동 생성	코드의 기능 및 명세를 이해하고 적절한 단위 테스트를 자동 생성함
		AI 기반 코드 완성	코드의 규칙과 구조에 따라 다음 코드를 예측하고 자동 완성을 제공함
		문서화 자동 생성	코드 주석과 문서를 자동으로 생성함
		데이터 접근 계층 자동 생성	데이터 모델을 기반으로 데이터 접근 계층을 생성함
	소프트웨어 문서화	문서의 지속적인 업데이트	코드 변경 사항을 감지하고 문서를 자동으로 업데이트함
		API의 자동 생성	코드에서 API를 추출하고 이를 기반으로 문서화 및 테스트 케이스를 생성함
		AI 기반 코드 설명	복잡한 코드를 분석하고 자연어로 설명을 제공함
		기술 문서 생성	전체 시스템의 기능을 이해하고 기술 문서를 작성함
		자연어 변환	코드의 기능을 자연어로 변환하여 비기술적 사용자도 이해할 수 있게 합
품질 보증과 유지보수	소프트웨어 테스팅 및 디버깅	테스트 케이스 자동 생성	코드를 분석하여 정적/동적 테스트 케이스를 자동으로 생성함
		버그 자동 탐지	코드에서 이상 패턴을 추출하여 버그를 자동으로 탐지함
		AI 지원 디버깅	코드를 파악하고 동작을 이해하여 디버깅 과정을 도움
		이상 탐지 자동화	실행 로그를 분석하여 이상 징후를 감지함
		AI를 통한 예측 유지보수	시스템 상태를 모니터링하고 유지보수 시점을 예측함
	리팩토링	자동 탐지 및 개선	코드 문제를 자동으로 탐지하고 개선함
		리팩토링을 위한 AI 기반 제안	코드 리팩토링을 위한 제안을 제공함
		성능 최적화를 통한 식별	성능 병목 지점을 식별하고 최적화 제안함
		레거시 코드베이스 현대화	오래된 코드를 분석하고 현대적인 코드로 리팩토링함
	협업 소프트웨어 개발	AI 지원 코드 협업	협업 도구를 통해 코드 리뷰 및 병합을 지원함
		충돌 해결 및 병합	충돌 패턴을 학습하여 자동 병합 전략 추천함으로 코드 충돌을 자동으로 해결하고 병합함
		지능형 코드 리뷰 지원	코드 리뷰를 도와주고 자동 피드백을 제공함으로 코드 품질 검사 및 개선을 제안함
		AI 지원 코드 리뷰	코드 품질과 표준을 검사하고 피드백을 제공 코드 스타일 및 규칙 위반 검사함
–	소프트웨어 보안	취약점 탐지 및 분석	코드를 스캔하고 잠재적인 보안 취약점을 탐지함, 정적 분석을 통해 보안 결함 탐지함
		AI 기반 침입 탐지	네트워크 트래픽을 분석하여 침입 시도를 탐지, 비정상적인 패턴을 실시간으로 감지하여 경고함
		자동화된 보안 테스트	자동화된 보안 테스트를 실행하여 취약점을 찾음
		악성코드 탐지 및 예방	파일 및 트래픽에서 악성코드를 탐지하고 차단함
		개인정보 보호 및 데이터 보호	데이터 사용을 모니터링하고 보호함
	소프트웨어 공학을 위한 자연어 처리	코드 요약을 통한 문서 생성	코드의 기능, 구조, 사용법 등을 요약하여 문서를 자동으로 생성함
		요구사항 분석 및 추출	소프트웨어 프로젝트의 요구사항을 문서에서 자동으로 식별하고 정리함
		코드에 대한 감정 분석	개발자나 사용자 리뷰, 피드백 등에서 코드에 대한 감정을 분석함
		사용자 자동 생성	소프트웨어 사용 설명서, 튜토리얼, 도움말 등을 자동으로 생성함
		코드 구조 개선을 위한 리팩토링	기존 코드의 기능을 유지하면서 구조를 개선하여 가독성, 유지보수성, 성능 등을 향상시킴
		이슈보고서 분류	깃허브 이슈보고서를 단일 및 다중 분류함

수행하는 연구 위주로 크게 2가지로 구분하여 설명한다. 먼저 소프트웨어 공학 문서 분류 자동화 연구와 소프트웨어 공학 문서 생성 연구로 구분할 수 있다. 본 원고에서는 생성 연구로는 테스트와 기타문서 산출물 생성에 관한 연구만 다룬다. 본 논문에서 소개하는 전반적인 연구에 대

한 요약은 표 3과 같다.

4.1 소프트웨어 공학 문서 분류 적용 연구

기계학습/딥러닝 기술의 발전으로 소프트웨어 산출물에서도 자동 분류 연구가 수행되고 있다. 최신 연구를 중심으로 정리한 이슈 분류 연구는 표 3에서 보이는 바와 같이 7개 정도를 소개한다.

2023년, Tikayat Ray et al.[4]은 BERT를 이용하여 항공우주 요구사항을 분류하는 기법을 제안하였다. 이들은 모델 기반 시스템 엔지니어링(Model-Based Systems Engineering, MBSE)이 자연어 요구사항의 모호성과 불일치로 인해 모델로 직접 변환하기 어렵다는 점을 지적했다. 이러한 요구사항들을 표준화하기 위해 저자들은 항공우주 요구사항을 세 가지 유형(디자인, 기능, 성능)으로 분류할 수 있는 aeroBERT-Classifier를 제안하고, 310개의 레이블링 된 항공우주 요구사항 코퍼스를 생성하였다. 실험을 위해 aeroBERT-Classifier, GPT-2, Bi-LSTM, bart-large-mnli 등 4가지 모델의 분류 결과를 비교하였으며, aeroBERT-Classifier가 0.83의 F1-score로 가장 좋은 성능을 보였다.

2023년, Gomes et al.[5]은 BERT와 TF-IDF를 피처 추출기로 사용하여 Free/Libre Open Source Software(FLOSS)에서 long-lived bug 예측 분류 기법을 제안하였다. long-lived bug를 예측하기 위해 Eclipse, Freedesktop, GCC, Gnome, Mozilla, WinHQ 등 6개의 FLOSS 프로젝트에서 버그 보고서를 수집하였고, k-NN, NB, NN, RF, SVM 기계학습 분류기를 구현하여 성능을 비교하였다. 실험 결과, BERT를 피처 추출기로 사용했을 때 SVM과 Random Forest에서 가장 좋은 성능을 보였으며, 버그 보고서로 추가 학습시킨 작은 아키텍처의 BERT가 추가 학습이 없는 큰 아키텍처의 BERT를 사용하는 것보다 더 높은 성능을 보였다.

2023년, Hadi et al.[6]은 모바일 애플리케이션 사용자 피드백 자동 분류를 위해 사전 훈련된 언어 모델을 활용한 분류 기법을 제안하였다. 이들은 6개의 데이터 셋에서 4개의 사전 훈련 언어 모델(BERT,

표 3 소프트웨어 개발 문서 자동 분류 및 생성 연구 요약

태스크	문헌	문헌 요약	년도
분류	Tikayat Ray et al.[4]	요구사항 문서 분류, 멀티클래스 분류 태스크 (설계, 기능, 성능) 사용모델: aeroBERT-Classifier, GPT-2, Bi-LSTM(GloVe), bart-large-mnli	2023
	Gomes et al.[5]	장기 지속 버그 예측, 이진 분류 태스크 사용모델: BERT와 TF-IDF를 특징 추출기로 사용하는 5개의 기계학습 모델	2023
	Hadi et al.[6]	앱 리뷰 분류, 이진/멀티클래스 분류 태스크 사용모델: 4개의 사전 훈련 모델 BERT, XLNet, RoBERTa, ALBERT과 이전 기계학습 기법	2023
	Kaur et al.[7]	요구사항 분류 사용모델 : BERT, BiLSTM, CNN	2023
	Zhifang et al.[8]	버그리포트 분류, 이진 분류 사용모델: FTNet와 6개의 비교 모델	2024
	Aracena et al.[9]	이슈보고서 분류, 다중클래스 분류(버그, 기능, 질문) 사용모델: gpt-3.5-turbo	2024
	Colavito et al.[10]	이슈보고서 분류, 멀티클래스 분류(버그, 향상, 질문, 문서) 사용모델: SETFIT, GPT3.5-turbo	2024
생성	Schäfer, et al.[11]	유닛 테스트 생성 사용모델: gpt3.5-turbo	2023
	Dakhel et al.[12]	뮤테이션 테스트 케이스 생성 사용모델: Codex와 llama-2-chat	2024
	Alagarsamy et al.[13]	텍스트-테스트 케이스 생성 사용모델: GPT-3.5	2024
	Aroraet al.[14]	테스트 시나리오 생성 사용모델: GPT-3.5	2024
	Palacio et al.[15]	코드 구문 구조와 모델 신뢰도 사이의 설명 생성 사용모델: GPT-3, CodeGen-NL, Multi-lang, Mono-lang의 다양한 사이즈 12개	2024
	Dhyani et al.[16]	API 문서 생성 사용모델: LLaMA 2-7B	2024
	Hassine et al.[17]	골 모델과 자연어 요구사항 간의 추적성 링크 생성 사용모델: GPT-3.5-turbo	2024

XLNet, RoBERTa, ALBERT)의 성능을 기계학습 모델로 구현한 이전연구 결과와 비교하였다. 실험 결과, 사전 훈련된 모델은 이진 및 다중 클래스 분류에서 모든 데이터 셋에 대해 높은 정확도를 보였고, 특히 ALBERT와 RoBERTa가 우수한 성능을 보였다. 또한, 사전 훈련에 도메인 특화 데이터인 앱 데이터를 포함하여 추가 학습하면 RoBERTa 커스텀 모델이 가장 높은 정확도를 보였다.

2023년, Kaur et al.[7]은 소프트웨어 요구사항을 파악하기 위해 BERT-BiCNN 기반 딥러닝 기법을 제안하였다. 이들은 소프트웨어 요구사항이 포함된 PROMISE 데이터 셋에서 제안된 모델을 평가하였다. 제안된 모델은 BERT 임베딩 레이어, 양방향 LSTM, CNN을 조합하였다. 실험 결과 BERT-BiCNN 모델은 정밀도 0.96, 재현율 0.93, F1-score 0.94로 가장 높은 성능을 보였다.

2024년, Zhifang et al.[8]은 깃허브 이슈 보고서를 분류하기 위해 전이 학습과 GitHub 숙련도와 프로젝트에 대한 친숙도를 파악하기 위해 제출자의 개인정보를 사용하여 GitHub 버그 리포트 분류 방법인 PIFTNet을 제안하였다. 실험 및 결과는 개인정보를 사용하기 전 이슈 리포트로 학습한 사전 학습 모델(FTNet)을 FastText, ATTBiLSTM, TextLSTM, TextANN, TextBiLSTM, TextCNN 등 6개의 모델과 비교 한 결과 FTNet이 0.8555의 F1-score로 가장 좋은 성능을 보였다. 이후 FTNet과 개인 정보를 사용하는 PIFTNet의 모델을 비교한 결과 PIFTNet이 약 0.3857 향상된 0.8588의 F1-score를 보이며 가장 좋은 성능을 보였다.

2024년, Aracena et al.[9]은 GitHub의 이슈 보고서를 버그, 기능, 질문으로 다중 클래스 분류하기 위해 gpt-3.5-turbo를 활용한 분류 기법을 제안하였다[9]. 이들은 학습 데이터의 제목과 본문을 프롬프트 일부로 또한 레이블을 예상 결과로 사용하여 gpt-3.5-turbo 모델을 fine-tuning 하였다. 실험 결과, 정밀도가 83.24%, 재현율이 82.87%, F-measure가 82.8%의 정확도를 보이며 GPT 기반 모델의 파인튜닝과 프롬프트를 통한 GitHub 이슈 분류 기법을 보였다.

2024년, Colavito et al.[10] GPT와 같은 대규모 언어 모델을 활용하여 이슈보고서 분류 자동화 기법을 제안하였다. 이들은 GPT-3.5-turbo를 활용하여 퓨 샷 및 제로 샷 설정에서 이슈 보고서 멀티클래스 분류 결과를 BERT 기반 모델인 SETFIT과 비교하였다. 실험 결과, GPT-3.5-turbo는 f1-micro에서 0.8155, f1-macro에서 0.8095를 보였으며, SETFIT은 f1-micro에서 0.8321, f1-macro에서 0.8246의 성능으로 소폭의 차이를 보였다. 이는 파인튜닝 된 SETFIT 모델과 달리 GPT-3.5-turbo는 파인튜닝 없이도 제로 샷 설정에서 비슷한 결과를 보였고, 레이블이 없는 데이터로도 이슈 보고서를 분류할 수 있는 능력을 보여준 것에 의의가 있다.

대규모 언어 모델을 활용한 분류 패러다임이 BERT 기반에서 GPT 기반으로 다양하게 시도되고 있다. 이는 GPT의 초거대 파라미터 수가 더 복잡한 패턴과 문맥을 학습하여 다양한 태스크에서 일반화 성능이 뛰어나고, 레이블링 데이터가 없어도 제로 샷과 퓨 샷에서 좋은 성능을 보이기 때문이다. 소프트웨어 공학 도메인에서 GPT 계열의 모델을 활용하여 분류 태스크를 수행하는 연구는 아직 초기 단계이기 때문에 향후 GPT 계열 모델을 활용한 분류 연구가 더 많이 수행될 것이라 예상된다.

4.2 소프트웨어 공학 문서 생성 적용 연구

생성형 AI의 발전으로 소프트웨어 공학 문서 자동 생성 연구가 수행되고 있다. 생성 연구로는 테스트 생성과 기타 산출물만 대상으로 표 3에서 보이는 바와 같이 7개 정도를 소개한다.

4.2.1 테스트 생성 연구

2023년, Schäfer et al.[11]은 유닛 테스트를 수동으로 생성하는 데 큰 노력이 필요함을 지적하였다. 이를 자동화하기 위해 함수명, 구현, 사용 예시를 포함한 프롬프트를 LLM에 제공하여 자동 유닛 테스트를 생성하는 기법인 nTestPilot를 제안하였다. 제안된 방법은 OpenAI의 gpt-3.5-turbo을 사용하여 25개의 npm 패키지에서 테스트를 수행하였고, 생성된 테스트는 70.2%의 문장 커버리지와 52.8%의 분기 커버리지를 달성하였으며, 기존의 Nessie 기술보다 우수한 성능을 보였다. 또한, 생성된 테스트 대부분은 기존 테스트와 50% 이하의 유사성을 보여 새로운 테스트 케이스의 독창성을 입증하였다.

2024년, Dakhel et al.[12]은 최근 코드의 대규모 언어 모델을 이용한 단위 테스트 생성이 주목받고 있으나, 생성된 테스트의 코드 커버리지가 버그 탐지 효율성을 보장하지 못하는 문제점을 지적했다. 이들은 LLM 중 Codex와 llama-2- chat을 사용하여 생성된 테스트 케이스의 효과를 개선하기 위해 MuTAP(Mutation Test case generation using Augmented Prompt)을 제안하였다. MuTAP은 제로 샷 및 퓨 샷 학습을 통해 테스트 케이스

를 생성하고, 오류 수정 후 뮤테이션 테스트로 평가하여 생존 돌연변이와 부적절한 테스트 케이스를 기반으로 초기 프롬프트를 증강한다. 이후 LLM에 다시 프롬프트 하여 새로운 테스트 케이스를 재생성한다. 다양한 벤치마크에서 MuTAP의 성능을 평가한 결과, 결함이 있는 코드 스니펫을 기존 방법보다 최대 28% 더 많이 감지하는 데 성공했으며, 합성 결함 코드에서 93.57%의 뮤테이션 스코어로 가장 좋은 결과를 보였다.

2024년, Alagarsamy et al.[13]은 테스트 주도 개발에 있어 기존의 코드 기반 자동 테스트 케이스 생성 방법이 요구사항을 입력으로 사용하는 테스트 주도 개발에 적용하는 데 한계가 있음을 지적했다. 이들은 GPT-3.5와 같은 대규모 언어 모델을 활용하여 효과적인 프롬프트 디자인을 통해 파인튜닝된 텍스트-테스트 케이스 생성 접근 방식을 제안하였다. 실험을 위해 다양한 오픈소스 소프트웨어 프로젝트를 대상으로 제안방법을 평가한 결과 생성된 테스트 케이스가 78.5%의 문법적 정확성, 67.09%의 요구사항 일치, 61.7%의 코드 커버리지를 달성하였다. 이는 기존 연구보다 월등한 성능을 보이며 파인튜닝과 프롬프트 디자인이 텍스트-테스트 케이스 생성 작업에 필수적임을 입증하였다.

2024년, Chetan Arora et al.[14]은 테스트 시나리오가 체계적으로 시스템을 테스트하는 데 중요하지만, 테스트 시나리오를 작성하는 것은 소프트웨어 기능과 도메인에 대한 깊은 이해와 노력이 필요함을 지적했다. 따라서 이들은 대규모 언어 모델과 검색 증강 생성(RAG)을 활용한 테스트 시나리오 생성을 자동화하는 RAGTAG 방법을 제안하였다. RAGTAG 접근법은 GPT-3.5를 퓨 샷 프롬프팅 기법과 결합하여 자연어 요구사항을 기반으로 프롬프트를 생성하고, RAG로 관련 문맥을 검색한 후, 대규모 언어 모델을 사용하여 테스트 시나리오를 생성한다. 제안 방법은 두 산업 프로젝트에서 평가되었으며, 전문가 인터뷰를 통해 RAGTAG의 유용성을 측정한 결과, 생성된 테스트 시나리오가 이해하기 쉽고 프로젝트 환경에서 실행 가능하며, 관련 기능의 다양한 측면을 커버하고 기본 요구사항과 잘 일치한다고 평가되며 RAGTAG의 효용성이 입증되었다.

4.2.2 기타 문서 생성 연구

2024년, Palacio et al.[15]은 대규모 언어 모델의 해석 가능성과 신뢰성을 높이기 위해 프로그래밍 언어의 구문 구조와 모델 신뢰도 사이의 관계를 기반으로 설명을 생성하는 ASTrust 기법을 제안하였다. 이 기법은 추상 구문 트리를 사용하여 생성된 코드를 설명하고, 모델 예측을 지역적 및 전역적으로 이해할 수 있도록 돕는다. 12개의 주요 대규모 언어 모델(GPT-3, CodeGen-NL, Multi-lang, Mono-lang)을 대상으로 실험을 진행한 결과, ASTrust가 모델 예측의 신뢰성을 이해하는 데 유용함을 입증하였다.

2024년, Dhyani et al.[16]은 생성형 AI를 활용하여 애플리케이션 프로그래밍 인터페이스(API) 사용 설명서 작성 자동화를 제안하였다. 이들은 다양한 기술 회사의 설명서와 산업 표준 설명서 데이터를 수집하여 Llama-2-7B-bf16-sharded 모델을 미세 조정하였다. 실험 결과를 비교하기 위해 모델을 파인튜닝 하기 전과 후의 결과를 비교하여 API 설명서 작성 정확성, 속도, 규모가 향상되었음을 입증하였다.

2024년, Hassine et al.[17]은 자연어 요구사항과 목표 모델의 목표 간 보안 추적성 링크를 생성하는 대규모 언어 모델 기법을 제안하였다. 제안된 방법은 GPT-3.5-turbo를 사용한 제로샷 학습 기법이며, 정교하게 설계된 프롬프트를 활용하여 정확도를 향상시켰다. 제안된 방법을 평가하기 위해 가상 인테리어 디자이너 애플리케이션의 목표를 설명하는 목표 모델과 보안 및 비보안 측면을 다루는 42개의 요구사항을 사용하였다. 실험 결과, 정확도 100%, 재현율 78.5%, F1-score 87.9%의 성과를 보였다.

5. 향후 소프트웨어 공학 태스크 자동화 연구 방향에 대한 논의

대규모 언어 모델을 활용한 소프트웨어 공학 태스크 중 분류 및 생성에 관한 연구를 보았다. 이렇게 검토한 문헌 연구를 바탕으로 향후 대규모 언어 모델을 활용한 소프트웨어 공학 태스크 자동화의 향후 연구 및 고려사항 3가지를 제안한다.

5.1 실사용을 반영한 범용적 이슈보고서 분류기 생성

본 원고에서 소개한 이슈 보고서 분류에 관한 연구[8, 9, 10]는 벤치마크 데이터셋을 사용하여 소수의 레이블을 대상으로 다중 클래스 분류를 수행하였다. 그러나 실제 GitHub 이슈 보고서와 레이블의 실사용은 다중 레이블과 프로젝트 특성이 고려된 종속적인 레이블이 사용되고 있다. 따라서 본 연구는 GitHub 이슈 보고서의 실사용과 괴리가 있다. 따라서 이슈 보고서 분류의 실사용을 반영하기 위해 분류 태스크를 이

진/멀티클래스에서 다중 레이블로 확장할 필요가 있다. 또한, 이슈 보고서 분류 태스크는 주로 BERT 기반의 분류기로 구현되었지만, 최근에는 GPT 기반의 모델로 분류를 수행하는 시도가 늘어나고 있다[22]. BERT 기반 분류기는 오픈소스 프로젝트마다 개별적인 분류기를 생성할 때 가장 좋은 성능을 보이기 때문에 프로젝트 종속적인 분류기를 생성해야 하는 단점이 있다. 반면, GPT 기반 모델로 분류기를 구축하면 검색 증강 검색(RAG), 프롬프트 엔지니어링, 퓨샷 러닝 등의 기술을 접목하여 프로젝트 범용적이면서도 프로젝트 세부 레이블을 고려할 수 있는 분류기 생성이 가능할 것으로 예상된다. 이를 통해 좀 더 효율적이고 적은 리소스로 레이블 분류기를 구현할 수 있을 것으로 기대되어 추가적인 연구가 필요하다.

5.2 대규모 언어 모델에 소프트웨어 공학 도메인 정보를 반영할 수 있는 기술 적용

일반적으로 대규모 언어 모델은 다양한 주제와 도메인을 포함하는 방대한 범용 데이터 셋으로 사전 학습된다. 이는 모델이 광범위한 일반 지식을 습득하게 하지만, 특정 도메인에 대한 깊이 있는 지식은 부족할 수 있다. 따라서 도메인 데이터로 대규모 사전 훈련 모델을 추가 학습함으로써 모델이 해당 도메인에서 자주 사용되는 용어, 개념, 문체 등을 학습하여 도메인 특화된 지식을 습득하도록 돕는다. 하지만 갈수록 언어 모델의 아키텍처와 규모가 커짐에 따라 파인튜닝을 하기가 쉽지 않다. 따라서 다양한 기술들을 접목할 필요가 있다. 몇 가지 예시를 들자면, 프롬프트 엔지니어링을 통해 N 샷 학습을 하여 언어 모델이 학습한 적이 없는 특정 태스크를 수행할 수 있게 도울 수 있고, 지시 증강 검색(RAG) 기술을 통해 언어 모델이 학습하지 않았거나 반영되지 않은 최신 정보를 내부 데이터베이스나 외부 검색을 통해 검색하여 언어 모델의 응답으로 포함할 수 있다. 또한 어댑티브 파인튜닝을 통해 모델의 특정 부분만 파라미터를 업데이트하는 기법과 파라미터를 경량화하는 기법인 양자화, 파라미터 효율적인 파인튜닝(PEFT), 낮은 순위 적응(LoRa) 등의 기술을 접목하여 소프트웨어 공학 지식을 가진 대규모 언어 모델로 발전시킬 수 있으며 소프트웨어 공학 태스크의 자동화 기술 성능을 높일 수 있을 거라 예상된다.

5.3 코드 기반의 산출물 외의 자연어, 다이어그램 기반의 산출물 자동 생성 연구

소프트웨어 개발 단계 중 개발 및 테스트 자동화에 관한 연구[11, 12, 13, 14]는 다양하게 이루어져 좋은 성과를 보이고 있지만, 요구사항 명세[18, 19] 및 설계 단계[20, 21]의 자동화 연구는 여전히 많은 도전 과제가 있다. 이는 자연어로 구성된 요구사항의 모호성과 불확실성, 비정형성으로 인한 어려움 때문이다. 또한 시스템 설계 단계는 시스템 설계 전문가의 의견과 풍부한 실무경험이 필요하고, 결과가 추상화 수준이 높은 산출물이기 때문에 생성에 어려움이 있다. 또한 코드 기반의 산출물은 코드 컴파일 여부와 테스트 성공 여부와 같은 명확하고 객관적인 평가 기준이 존재하는 반면, 자연어와 다이어그램의 평가 기준은 명확성, 일관성, 적절성 등 평가 기준이 주관적이고 모호한 어려움이 있다. 따라서 소프트웨어 공학의 요구분석 및 설계 자동화를 위한 연구는 아직 낮은 진척도를 보이고 있으며, 이는 향후 도전해볼 만한 유망한 연구 분야로 예상된다.

6. 결 론

본 원고를 통해 대규모 언어 모델 개념을 간단히 소개하고, 소프트웨어 공학 문서 분류와 생성 연구의 추세를 정리하였다. 또한 연구 추세의 분석에 기반하여 소프트웨어 개발 문서 분류와 생성 방향에 대하여 논의하였다. 우리는 도전적인 향후 연구에 대한 비전을 가지고 대규모 언어 모델을 사용하여 이슈 보고서 분류와 API 문서 자동 생성을 효과적으로 할 수 있는 LLM4SE, AI4SE에 관한 연구를 진행하고 있다. 우리는 해당 원고가 소프트웨어 공학 문서 자동 분류와 생성에 관심 있는 다른 연구자에게 유익한 참고 자료가 되기를 기대한다.

참고문헌

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[2] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[3] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

[4] Tikayat Ray, A., Cole, B. F., Pinon Fischer, O. J., White,

R. T., & Mavris, D. N. (2023). aerobert-classifier: Classification of aerospace requirements using bert. Aerospace, 10(3), 279.

[5] Gomes, L., da Silva Torres, R., & Côrtes, M. L. (2023). BERT-and TF-IDF-based feature extraction for long-lived bug prediction in FLOSS: a comparative study. Information and Software Technology, 160, 107217.

[6] Hadi, M. A., & Fard, F. H. (2023). Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews. Empirical Software Engineering, 28(4), 88.

[7] Kaur, K., & Kaur, P. (2023). Improving BERT model for requirements classification by bidirectional LSTM-CNN deep model. Computers and Electrical Engineering, 108, 108699.

[8] Zhifang, L., Kun, W., Qi, Z., Shengzong, L., Yan, Z., & Jianbiao, H. (2024). Classification of open source software bug report based on transfer learning. Expert Systems, 41(5), e13184.

[9] Aracena, G., Luster, K., Santos, F., Steinmacher, I., & Gerosa, M. A. (2024). Applying Large Language Models API to Issue Classification Problem. arXiv preprint arXiv:2401.04637.

[10] Colavito, G., Lanubile, F., Novielli, N., & Quaranta, L. (2024, April). Leveraging GPT-like LLMs to Automate Issue Labeling. In 2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR) (pp. 469-480). IEEE.

[11] Schäfer, M., Nadi, S., Eghbali, A., & Tip, F. (2023). An empirical evaluation of using large language models for automated unit test generation. IEEE Transactions on Software Engineering.

[12] Dakhel, A. M., Nikanjam, A., Majdinasab, V., Khomh, F., & Desmarais, M. C. (2024). Effective test generation using pre-trained large language models and mutation testing. Information and Software Technology, 171, 107468.

[13] Alagarsamy, S., Tantithamthavorn, C., Arora, C., & Aleti, A. (2024). Enhancing Large Language Models for Text-to-Testcase Generation. arXiv preprint arXiv: 2402.11910.

[14] Arora, C., Herda, T., & Homm, V. (2024). Generating Test Scenarios from NL Requirements using Retrieval-Augmented LLMs: An Industrial Study. arXiv preprint arXiv:2404.12772.

[15] Palacio, D. N., Rodriguez-Cardenas, D., Velasco, A., Khati, D., Moran, K., & Poshyvanyk, D. (2024). Towards More Trustworthy and Interpretable LLMs for Code through Syntax-Grounded Explanations. arXiv preprint arXiv:2407.08983.

[16] Dhyani, P., Nautiyal, S., Negi, A., Dhyani, S., & Chaudhary, P. (2024, February). Automated API Docs Generator using Generative AI. In 2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS) (pp. 1-6). IEEE.

[17] Hassine, J. (2024, June). An llm-based approach to recover traceability links between security requirements and goal models. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (pp. 643-651).

[18] Budake, R., Bhoite, S., & Kharade, K. (2023, November). A study of AI-based techniques for requirement analysis in software engineering. In AIP Conference Proceedings (Vol. 2946, No. 1). AIP Publishing

[19] Arora, C., Grundy, J., & Abdelrazek, M. (2024). Advancing requirements engineering through generative ai: Assessing the role of llms. In Generative AI for Effective Software Development (pp. 129-148). Cham: Springer Nature Switzerland.

[20] Cámara, J., Troya, J., Burgueño, L., & Vallecillo, A. (2023). On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. Software and Systems Modeling, 22(3), 781-793.

[21] Alshareef, A., Keller, N., Carbo, P., & Zeigler, B. P. (2023, December). Generative AI with Modeling and Simulation of Activity and Flow-Based Diagrams. In International Conference on Simulation Tools and Techniques (pp. 95-109). Cham: Springer Nature Switzerland.

[22] Heo, J., Kwon, G., Kwak, C., & Lee, S. (2024). A Comparison of Pretrained Models for Classifying Issue Reports. IEEE Access.

약 력

허 주 은

2022 경상국립대학교 항공우주 및 소프트웨어공학 전공 (학사)

2024 경상국립대학교 AI융합공학과 (석사)

2024 ~ 현재 경상국립대학교 AI융합공학과 (박사과정)

관심분야 : 소프트웨어 공학, 자연어 처리, 대규모 언어 모델

Email : juandeun@gnu.ac.kr

이 선 아

1997 이화여자대학교 전산학(학사)
1999 이화여자대학교 전산학(석사)
2005 카네기멜론대학교 소프트웨어공학(석사)
2013 KAIST 전산학(박사)
1999~2006 삼성전자 선임/책임연구원.
2013~2015 KAIST 연구 조교수.
2016~현재 경상국립대학교 소프트웨어공학과 / AI융합공학과 부교수.
관심분야 : Software Architecture & Software Repository, Mining & Recommendation System & Software Evolution
Email : saleese@gnu.ac.kr
http://orcid.org/0000-0002-2004-2924