LLM의 입력 길이 제한 조건 하에서 패치 경량화와 문맥 활용 기법*

정은서[1], 아슬란 압디나비예프[1], 홍수지[2], 이병정[1]
[1]서울시립대학교 컴퓨터과학과, [2]서울시립대학교 인공지능학과
{eunseoj, aslan, sujiggum, bjlee}@uos.ac.kr

Leveraging Patch Lightweighting and Context under Constraint of Input Size of LLM

Eunseo Jung[1], Abdinabiev Aslan Safarovich[1], Suji Hong[2], Byungjeong Lee[1]
[1]Department of Computer Science, [2]Department of Artificial Intelligence, University of Seoul

요 약

프로그램 자동 정정(Automated Program Repair, APR)은 소프트웨어 개발 과정에서 발생하는 버그나 에러를 자동으로 정정하는 기술로, 최근 대규모 언어 모델(Large Language Model, LLM)의 발전과 함께 적극적으로 활용되고 있다. 그러나 대부분의 LLM은 입력 길이(토큰 수)에 제한이 존재하기 때문에, 긴 메서드나 문맥 의존성이 높은 복잡한 코드를 효과적으로 정정하는 데 한계가 있다. 본 연구는 이러한 한계를 극복하기 위해, 버그가 포함된 메서드와 주변 문맥 메서드를 LLM의 토큰 제한에 맞게 경량화하여 입력 프롬프트를 구성하는 방법을 제안한다. 이 과정에서 임베딩 기반 유사도를 활용해 문맥 관련도가 높은 메서드를 추출하고, 경량화를 통해 핵심 정보를 유지하는 입력을 구성한다. 이를 통해 본 연구는 LLM 기반 APR의 입력 제약 문제를 실질적으로 극복할 수 있는 효과적인 프롬프트 구성 방안을 제시하며, 향후 다양한 코드 환경에 적용 가능한 확장성 있는 APR 기술의 기반을 마련한다.

1. 서론

소프트웨어 개발 과정에서 발생하는 버그는 시스템의 안정성과 신뢰성에 큰 영향을 미치며, 이를 해결하기 위한 프로그램 자동 정정(Automated Program Repair, APR) 연구의 중요성이 점점 커지고 있다. 특히 대규모 언어 모델(Large Language Model, LLM)의 등장 및 발전으로, APR에 적용이 활발해졌지만, 모델의 토큰 제한으로 인해, 긴 메서드나 의존성이 높은 복잡한 코드를 효과적으로 정정하는 데에는 여전히 한계가 존재한다. 따라서, 본 연구는 이러한 LLM의 한계를 극복하기 위해 사용하는 LLM의 토큰 제한에 맞춰 버기 메서드와 문맥 메서드를 효과적으로 경량화하여 입력 프롬프트를 구성하는 방법을 제안한다. 메서드 단위의 유사도를 통해 유사한 문맥 메서드를 선정하고, LLM의 토큰 제한에 맞게 메서드를 경량화한 다음 후보패치를 생성하고, 올바르게 버그가 정정되었는지 검증한다. 이를 통해 단일 라인이 아닌, 여러 줄에 거친 버그이거나 여러 위치에 분포하는 버그 즉 복합적인 버그(Complicated Bug)를 정정하는 것이 목적이다. 본 연구는 크게 세가지 측면에서 그 유용성을 검증하고자 한다. **(1) 긴 메서드 버그가 정정되었는지**, **(2) 다양한 문맥 메서드 정보가 활용되었는지**, **(3) 경량화 방식에 따라 정정 결과가 차이가 발생하는지를** 실험을 통해 분석하였다. 이러한 접근방법을 통해, 본 연구는 LLM의 한계를 극복하고, 실제 소프트웨어 환경에서 안정적인 성능을 유지할 수 있는 현실적인 APR 기술을 제안한다.

2. 관련 연구

RAP-Gen[1]은 버그 코드가 입력되면 어휘적 및 의미적 유사성을 고려하여 독립적인 코드베이스에서 가장 적절한 버그-수정 쌍을 검색한다. 이후 검색된 쌍과 버그 코드를 활용하여 미세 조정한 CodeT5[2] 모델을 통해 패치를 생성한다. 하지만 검색된 코드 쌍에 따라 성능이 좌우되며, 검색된 정보가 LLM의 입력 토큰 제한을 초과할 경우 정보 손실이 발생할 수 있다는 한계가 있다.

ITER[3]는 버그 위치 확인, 패치 생성, 패치 검증을 여러 번 반복하여 여러 위치에 분포한 버그를 수정한다. 하지만 사용한 모델의 토큰 제한으로 인해 최대 384개의 토큰을 입력하고 최대 76개의 토큰을 생성하도록 설정하고 실험을 진행하였고, 이로 인해 복잡하거나 길이가 긴 코드 정정에 필요한 문맥 정보가 충분히 반영되지 않을 수 있다.

AlphaRepair[4]는 버그가 있는 라인을 <mask>토큰으로 대체한 후 주변 문맥 정보를 통해 CodeBERT[5]가 적절한 코드를 채워 넣는 방식으로 패치를 생성한다. 모델은 추가적인 학습 없이 (zero-shot) 사전 학습된 상태로 활용되며, 다양한 후보 패치를 탐색하기 위해 Complete Mask, Partial Mask, Template Mask의 세 가지 마스킹 전략을 사용한다. 이러한 방식으로 최대 5,000개의 후보 패치를 생성한 뒤, CodeBERT의 확률 기반 점수로 재정렬하여 우선순위가 높은 패치부터 검증한다.

3. 패치 경량화와 문맥 정보

3.1. 개요

본 연구의 전체 개요는 그림1과 같고, 유사도가 높은 문맥 메서드 추출, 메서드 경량화, 패치 생성, 패치 복원 그리고 패치 최적화와 평가를 거쳐 올바르게 버그가 정정되었는지 검증하는 5단계로 구성되어 있다. 먼저 버그가 있는 파일이 입력되면, 버그가 있는 메서드 즉 버기 메서드(Buggy Method)와 기타 메서드들(Methods)로 분류한다. 그 중 버기 메서드와 가장 유사한 문맥 메서드 5개를 추출한다. 이후 길이가 긴 버기 메서드와 문맥 메서드는 모델의 토큰 제한에 맞게 경량화된다. 경량화된 버기 메서드(Lightweight Buggy Method)와 경량화된 문맥 메서드(Lightweight Context Method)는 쌍을 이루어 입력 프롬프트를 구성하고, 경량화 구조를 학습한 미세조정된 LLM을 통해 경량화된 후보패치(Lightweight Candidate Patch)를 생성한다. 생성된 후보패치는 패치 복원(Patch Reconstruction)을 통해 온전한 메서드 형태를 갖춘 후보 패치(Original Candidate Patch)로 복원[6]된다. 복원된 후보패치는 패치 필터링, 패치 순위화, 패치 조합의 과정으로 이루어진 패치 최적화를 거친 후 패치 평가를 통해 문법적으로 오류가 없고(Compilable), 테스트 케이스 또한 모

* 본 연구성과물은 2024년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (No. RS-2024-00465887)
본 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (NO. RS-2022-NR068754)

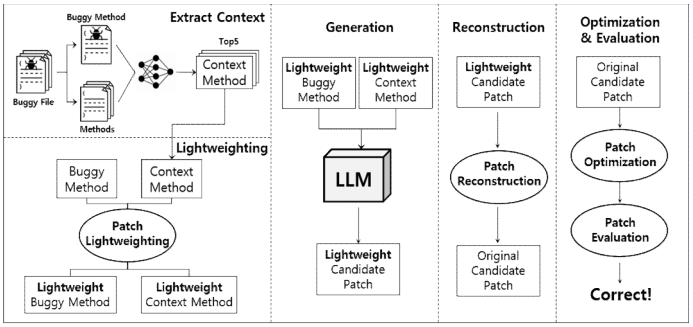

그림 1 전체 개요

두 통과하는지(Plausible) 확인한 후 최종적으로 개발자가 직접 올바르게(Correct) 버그가 수정되었는지 확인한다.

3.2. 패치 경량화

버기 메서드와 문맥 메서드의 길이가 사용하는 LLM의 토큰 제한을 넘는 경우 메서드 내에 필요한 정보들만 추출되어 메서드를 구성할 수 있도록 경량화한다. 경량화[6]는 버그가 있는 라인 즉 버기 라인(Buggy Line)을 기준으로, 각 라인에 대해 줄 번호 간의 차이를 기반으로 한 거리 점수와 라인별 임베딩된 벡터 간의 코사인 유사도에 기반한 유사도 점수를 구한 후 두 점수를 종합하여 최종적으로 관련도 점수를 얻는다. 관련도 점수를 기반으로 점수가 낮은 라인부터 순차적으로 제거하고, LLM의 토큰 제한 내에서 입력 가능한 길이로 메서드를 구성한다. 이러한 방식은 정정 대상 코드와 관련도가 높은 문맥적 정보와 의미적 연관성을 고려하여 정보를 선별하기 때문에 효과적인 정정 성능을 유지할 수 있는 구조적 강점을 갖는다. 최종적으로 경량화된 메서드는 LLM의 토큰 제한을 만족하면서도, 정정에 필요한 핵심 문맥을 최대한 보존하여 입력된다.

3.3. 패치 문맥

버그를 수정하는데 필요한 올바른 코드의 상당 부분은 동일한 파일 내에 존재하는 경우가 많다. 이에 따라 버그가 있는 파일 내에서 버기 메서드와 유사도가 높은 문맥 메서드들을 추출하여 모델에 함께 입력한다. 이를 위해 먼저 버기 메서드를 포함한 모든 메서드를 메서드 단위로 임베딩한다. 만약 임베딩 모델이 처리 가능한 범위 이상의 토큰으로 구성된 메서드인 경우 슬라이딩 윈도우 기법을 적용하여 메서드를 분할한 후 임베딩을 진행한다. 이때 윈도우 크기는 모델의 최대 입력 토큰 수로 설정하고, 윈도우 간에는 100토큰의 중첩(overlap)을 적용하여 문맥의 단절을 최소화한다. 이를 통해 모델이 코드의 연속적인 흐름을 보다 잘 이해할 수 있도록 한다. 예를 들면 512개의 토큰 한계를 가진 모델을 사용하여 1,000개의 토큰을 가진 메서드를 임베딩하는 경우 [0-512], [412-924], [824-1000] 으로 메서드를 나누어 임베딩 한 후 임베딩 값의 평균값을 최종 메서드의 임베딩 값으로 사용한다. 이를 통해 버기 메서드와 유사도가 높은 5개의 문맥 메서드를 추출하고 하나의 문맥 메서드와 버기 메서드를 쌍으로 입력 프롬프트를 구성하여 모델에 입력한다. 문맥 메서드를 경량화해야 하는 경우 문맥 메서드 안에서 버기 라인과 가장 유사한 라인을 찾아 문맥 메서드의 버기 라인으로 지정하고 그 라인을 기준으로 문맥 메서드를 경량화한다. 이러한 방식으로 버기 메서드 범위 밖의 문맥 메서드 정보를 모델에 함께 제공하여, 올바르게 버기 메서드를 수정할 수 있도록 한다.

4. 실험

본 연구는 입력이 512개 토큰까지 가능한 CodeBERT[5] 모델을 패치 생성 및 임베딩 과정에서 사용하여 실험을 진행하였다. 이에 따라 버기 메서드는 최대 300개, 문맥 메서드는 최대 200개 토큰 범위내로 경량화되도록 설정했다. 각 버기 메서드에 대해 유사도가 높은 문맥 메서드 5개를 각각 독립적으로 조합하여 입력 프롬프트를 구성하였고, 각 조합별로 100개의 후보 패치를 생성하였다. 즉, 하나의 버기 메서드에 대해 5개의 서로 다른 문맥 메서드를 적용하여 총 5개의 프롬프트를 만들고, 각 프롬프트에서 100개의 후보 패치를 생성함으로써 최종적으로 버기 메서드당 500개의 후보 패치를 생성하였다. 모델이 경량화된 메서드 형태를 이해할 수 있도록 Bugs2fix[7] 데이터셋을 통해 경량화된 버기 메서드와 문맥 메서드로 이루어진 입력 프롬프트와 경량화된 정정 메서드 쌍을 구성하여 모델을 미세조정하였다. 검증은 Defects4J[8] 데이터셋을 사용하였으며 버그 발생 위치를 제공하고 실험을 진행하였다. Bugs2fix와 Defects4J 데이터 모두 자바 코드 데이터이다. 실험은 NVIDIA RTX A6000 그래픽 카드, Intel Xeon Gold 6226R 2.9GHz CPU, 512GB RAM, Ubuntu 22.04 LTS 환경에서 진행되었다.

4.1. 긴 메서드 정정 결과

RQ1. 길이가 긴 메서드 버그가 정정되었는가?

본 연구는 Defects4J 데이터셋 중 버기 메서드의 길이가 사용한 모델인 CodeBERT의 입력 한도인 512개 토큰을 초과하는 124개의 버그를 선정하여 실험을 수행하였다. ITER는 Defects4J의 10개 프로젝트, 총 476개 버그만을 선정하여 테스트 하였기 때문에 ITER와 동일하게 476개 버그 중 긴 메서드 버그인 91개만을 대상으로 결과를 비교했다. 본 연구와 관련 연구인 ITER, RAP-Gen 그리고 AlphaRepair의 긴 메서드 정정 결과는 표 1과 같다. 본 연구와 AlphaRepair는 최대 15개의 버그를 정정하였으나 AlphaRepair가 최대 5,000개의 후보패치, ITER가 최대 1,000개의 후보패치 생성을 통해 버그를 정정한 것과 비교하여 본 연구는 상대적으로 적은 500개의 후보패치를 생성하여 버그를 정정했다. 정정한 버그의 평균 토큰 길이는 본 연구가 728개, ITER는 1,217개, RAP-Gen은 630개, AlphaRepair는 1,160개이다. ITER와 AlphaRepair에는 4,500 토큰이 넘는 버그를 정정한 사례가 1건 포함되어 있어 평균 토큰 길이가 상대적으로 높게 나타났다. 본 연구와 기존 연구에서 정정한 버그의 분포 및 사용한 모델의 정보는 그림 2와 같다. 본 연구는 다른 연구에서 정정하지 못한 3개의 버그를 올바르게 정정하였다. 해당 버그들은 여러 위치에 걸쳐 발생한 버그를 교체와 삭제 등의 다양한 수정 패턴을 동시에 적용하여 정정해야 하거나 특정 문맥 메서드에만 포함된 토큰 정보가 제공되어야만 정정이 가능한 버그였다. 이러한 특징을 통해 표1에서 복합적인 버그 즉, 여러 줄에 걸쳐있거나 여러 위치에 분포하는 버그의 정정 개수는 본 연구가 가장 많은 6개를 정정한 것을 확인할 수 있다.

4.2. 다양한 문맥 정보

RQ2. 다양한 문맥 정보가 유용한가?

다양한 문맥 메서드를 활용하는 것이 버그 정정에 미치는 효과는 표 2와 같다. 표2는 올바르게 정정된 패치가 처음으로 발견된 문맥 메서드 조합이 몇 번째 문맥 메서드였는지 나타낸다. 실험 결과 평균적으로 1.3번째 문맥 메서드 쌍에서 버그가 정정되었다. 그 중 특히 Closure14는 특정 토큰이 반드시 제공되어야만 정정이 가능한 버그

표 1 정정한 버그 정보

	This Study	ITER	RAP-Gen	Alpha Repair
패치 개수	500	1,000	100	5,000
버그 정정 개수 (124/91)	**15/12**	-/8	10/7	**15/12**
평균 토큰 길이	728	**1,217**	630	1,160
복합적인 버그 정정 개수	**6**	4	5	4

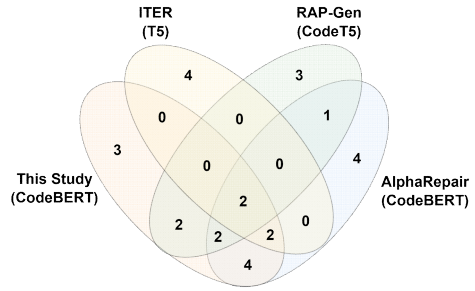

그림 2 정정한 버그 분포

였으며, 해당 토큰은 다섯 번째 문맥 메서드에만 포함되어 있었다. 이러한 결과는 단일 버기 메서드 또는 가장 유사한 문맥 메서드를 제공하는 것보다 파일 내 다양한 정보를 함께 제공하는 것이 더 많은 버그를 정정하는데 효과적임을 나타낸다.

4.3. 패치 경량화 방식

RQ3. 경량화 방식에 따라 정정 결과가 차이가 발생하는가?

패치 경량화는 버기 라인을 기준으로 거리 점수와 유사도 점수를 종합하여 얻은 관련도 점수를 통해 수행된다. 따라서 관련도 점수의 효과성을 검증하기 위해 거리 점수만을 적용한 경우, 유사도 점수만을 적용한 경우 그리고 두 요소를 각각 50%씩 반영한 관련도 점수를 적용한 경우로 나누어 경량화를 수행하고, 각 방식에서 실제로 긴 버기 메서드 정정이 가능한지 실험하였다. 실험 결과는 표3과 같다. 실험 결과, 유사도 점수만을 적용한 경우 해당 방식으로만 정정할 수 있는 긴 메서드는 1개이지만, 거리 점수나 관련도 점수가 수정할 수 있는 버그를 수정하지 못한 메서드는 3개였다. 또한 거리 점수만 적용하였을 때 고유하게 정정할 수 있는 메서드는 발견하지 못했으며, 유사도 점수와 관련도 점수를 적용하면 정정할 수 있는 메서드를 2개 정정하지 못하였다. 이를 통해 거리 점수나 유사도 점수를 종합하지 않고 단독으로 적용할 경우 한계가 존재함을 나타내고, 이를 종합한 관련도 점수를 기반으로 경량화하는 것이 효과적임을 확인할 수 있다.

5. 토의

올바르게 버그를 정정하지 못한 후보패치를 분석한 결과, 버그가 발생한 위치에 적절한 코드 삽입이 이루어지지 않아 정정에 실패한 경우가 다수 확인되었다. 적절한 코드 삽입을 위해서는 코드 앞뒤 문맥 파악은 물론, 에러 발생 원인, 삽입이 필요한 이유 등 보다 확장된 문맥 정보가 필요함을 의미한다. 따라서 더 많은 복합적인 버그 정정을 위해서는 삽입 중심 정정이 이루어질 수 있도록 보완이 필요하다. 또한 문맥 메서드에서 버그 정정을 위한 핵심 식별자 예를 들면, 메서드명, 식별자 등이 추가적으로 제공되었음에도 정정에 실패한 사례도 있었다. 특히 긴 식별자 명의 경우, 토크나이저가 이를 여러 서브워드로 분해하여 인식하게 되고, 이로 인해 모델이 해당 식별자명을 하나의 단위로 인식하지 못했을 가능성이 있다. 이는 서브워드 수준에서의 정보 손실 문제로 해석될 수 있으며, 향후에는 주요 식별자에 대한 가중치 부여 또는 명시적인 강조 방법을 통해 모델이 프롬프트 내 핵심 정보를 효과적으로 활용할 수 있도록 개선할 필요가 있다.

표 2 버그가 최초 정정된 문맥 메서드 위치 분포

문맥 메서드 순번	버그 정정 개수
첫 번째 문맥	13
두 번째 문맥	1
다섯 번째 문맥	1

표 3 관련도 점수의 효과성

	정정 성공	정정 실패	고유한 정정
거리+유사도 기반	17	1	1
유사도 기반	15	3	1
거리 기반	16	2	0

6. 결론

본 연구에서는 길이가 긴 메서드를 토큰 제한이 있는 LLM을 사용하여 버그 정정을 시도할 경우 발생할 수 있는 한계점을 경량화와 추가적인 문맥 메서드 제공을 통해 더 많은 버그를 정정할 수 있음을 실험을 통해 확인하였다. 본 연구 방법을 통해 한정된 입력 길이 내에서 효과적인 입력 프롬프트를 구성하여 적은 패치로도 길이가 긴 복합적인 버그를 정정할 수 있음을 확인했다. 또한 메서드를 경량화할 때 거리점수와 유사도를 종합한 관련도 점수를 적용하여 경량화할 경우 적은 토큰이라도 더 의미있는 정보만을 포함하여 더 많은 버그 정정이 가능함을 확인하였다. 이를 통해, 본 연구는 LLM의 토큰 제한이라는 구조적인 한계를 경량화와 문맥 정보 확장을 통해 효과적으로 극복할 수 있음을 실험적으로 입증하였고, 이는 다양한 LLM을 사용하는 APR 연구에서 그 실용성과 확장 가능성을 높이는 데 기여할 수 있다. 향후 연구에서는 적절한 코드 삽입 및 식별자, 메서드명 등에 대한 인식 정확도를 보완함으로써, 모델이 프롬프트 내 핵심 정보를 효과적으로 활용할 수 있도록 개선할 계획이다.

참고문헌

[1] W. Wang, Y. Wang, S. Joty and C.H. Hoi, "RAP-Gen: Retrieval-Augmented Patch Generation with CodeT5 for Automatic Program Repair," in Proc. of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp.146-158, 2023.

[2] Y. Wang, W. Wang, S. Joty and S. Hoi, "CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation," in Proc. of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp.8696-8708, 2021.

[3] H. Ye and M. Monperrus, "ITER: Iterative Neural Repair for Multi-Location Patches," in Proc. of the 46th IEEE/ACM International Conference on Software Engineering (ICSE), pp.1-13, 2024.

[4] C. Xia and L. Zhang, "Less Training, More Repairing Please: Revisiting Automated Program Repair via Zero-shot Learning," in Proc. of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), pp.959-971, 2022.

[5] Z. Feng et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," Findings of the Association for Computational Linguistics: EMNLP, pp.1536-1547, 2020.

[6] E. Jung, A. Safarovich and B. Lee, "Utilizing Patch Lightweighting and Reconstruction to Handle Token Length Issues in LLM based Automatic Program Repair," in Proc. of the Korea Software Congress, pp.381-383, 2024.

[7] M. Tufano et al, "An empirical study on learning bug-fixing patches in the wild via neural machine translation", ACM Transactions on Software Engineering and Methodology (TOSEM), pp.1-67, 2019.

[8] https://github.com/rjust/defects4j