LLM 기반 프로그램 자동 정정에서 토큰 길이 문제 처리를 위한 패치 경량화와 복원 활용

정은서, 아슬란 압디나비예프, 이병정
서울시립대학교 컴퓨터과학과
{eunseoj, aslan, bjlee}@uos.ac.kr

Utilizing Patch Lightweighting and Reconstruction to Handle Token Length Issues in LLM based Automatic Program Repair

Eunseo Jung, Abdinabiev Aslan Safarovich, Byungjeong Lee
Department of Computer Science, University of Seoul

요 약

프로그램 자동 정정 연구에서 거대언어모델(LLM)은 매우 중요한 역할을 한다. 하지만 토큰 제한 문제로 인해 LLM이 처리할 수 있는 토큰 개수의 한계를 넘는 경우 올바르게 버그나 에러를 정정하지 못하는 문제가 발생한다. 따라서 해당 연구에서는 LLM에서 처리할 수 있는 토큰 개수를 기반으로 하여 패치 경량화와 패치 복원 방법을 통해 더 많은 버그를 정정하고자 한다. 버그가 있는 메서드 중 토큰 개수가 모델이 처리 가능한 범위를 넘는 메서드는 패치 경량화 방법을 사용하여 LLM이 처리 가능한 범위의 토큰을 가질 수 있도록 한다. 그 다음 경량화된 메서드의 형태를 학습한 Fine-tuning된 모델을 통해 경량화된 후보패치를 생성하고, 이를 다시 온전한 메서드의 형태를 갖출 수 있도록 패치 복원 과정을 거친다. 이렇게 최종 생성된 패치는 버그를 적절하게 수정한 내용을 포함하고 온전한 메서드의 형태 또한 유지하고 있으므로, 이러한 방법을 통해 더 많은 버그를 수정할 수 있다.

1. 서 론[1]

소프트웨어 프로그램에 작성되는 코드는 점점 방대해지고 있으며, 이러한 방대한 코드 속에서 발생하는 버그를 사람이 직접 찾아서 수정하는 것은 엄청난 시간과 자원이 든다. 따라서 코드의 버그를 자동으로 정정하는 프로그램 자동 정정 연구(Automatic Program Repair, APR)의 중요성은 점점 커지고 있다. APR연구에서 거대언어모델(Large Language Model, LLM)은 연구 결과에 큰 영향을 미치기 때문에 적절한 모델을 선택하는 것은 매우 중요하다. 하지만 모델이 처리 가능한 토큰 개수를 초과하는 방대한 양의 코드를 투입하게 되면 모델의 성능을 최대한으로 활용하기 어렵다. 따라서 LLM이 처리 가능한 토큰 범위 내에서 버그를 수정하기 위한 가장 중요한 정보만을 선별해서 모델에 투입하는 것은 매우 중요하다. 따라서 본 연구에서는 패치 경량화와 패치 복원 방법을 통해 LLM의 토큰 제한 문제를 극복하여, 하나 또는 그 이상의 위치에서 발생하는 한 라인 이상의 멀티 청크 버그(multi-chunk bug)를 동시에 수정하고자 한다.

2. 관련 연구

MCRepair[1]는 버그가 있는 메서드를 buggy block으로 구성하여 Fine-tuning한 CodeBERT[2]모델을 통해 후보패치 생성 후 패치 최적화를 통해 버그를 수정한다. 하지만 CodeBERT가 처리 가능한 토큰 개수를 고려하지 않고 버그를 수정하여 메서드의 토큰 개수가 모델이 처리할 수 있는 개수보다 많은 다수의 메서드들을 올바르게 수정하지 못했다.

RepairLLaMA[3]는 CodeLlama-7B[4]모델에 LoRA를 함께 사용하여 Fine-tuning을 진행 후 패치를 생성하고, 최적의 코드 투입과 코드 생성 방식의 조합을 통해 메서드 단위의 버그를 수정하는 것을 목표로 한다. 하지만 Fine-tuning 진행 시 투입 데이터와 생성 데이터의 총합이 1024개를 넘지 않은 데이터 쌍만을 사용하였다.

ITER[5]는 T5[6]모델을 이용하여 버그 위치 확인, 패치 생성, 패치 검증을 여러 번 반복하여 여러 위치에 분포한 버그를 수정한다. 하지만 사용한 모델의 토큰 제한으로 인해 최대 384개의 토큰을 투입하고 최대 76개의 토큰을 생성하도록 설정하고 실험을 진행하였다.

3. LLM 토큰 길이 문제 처리

3.1. 개 요

본 연구의 전체적인 과정의 개요는 그림1과 같다. 실험 과정은 크게 패치 경량화, 패치 생성, 패치 복원, 패치 최적화, 패치 평가의 단계로 이루어져 있다. 먼저 버그가 있는 파일이 투입되면 버그가 발생하는 메서드, 즉 버기 메서드(Original Buggy Method, OBM)를 추출하고 메서드의 토큰 개수를 확인한다. 만약 버기 메서드(Short OBM)의 토큰 개수가 LLM이 처리할 수 있는 최대 토큰 개수를 넘지 않을 경우 패치 경량화 방법을 거치지 않고 패치 생성 단계를 진행한다. 만약 버기 메서드(Long OBM)가 모델이 처리할 수 있는 토큰 개수를 넘는다면, 패치 경량화 방법을 적용하여 모델이 처리 가능한 토큰 개수 이하가 되도록 버기 메서드(Lightweight Buggy Method, LWBM)를 구성하고 패치 생성 단계를 진행한다. 패치 생성에서 사용할 모델은 경량화된 버기 메서드(LWBM)와 경량화된 수정된 메서드(Lightweight Fixed Method, LWFM) 쌍으로 이루어진 대용량의 데이터셋을 이용하여 LLM을 Fine-tuning을 한 후 사용한다. 이러한 경량화된 메서드의 패턴을 익힌 LLM을 통해 후보패치(Candidate Patch)를 생성한다. 이후 경량화된 후보패치(Lightweight Candiate Patch, LWCP)는 패치 복원 과정을 통해 후보패치 내용과 본래의 메서드 형태를 모두 갖출 수 있도록 수정하고, 패치 복원까지 완료된 후보패치(Original Candiate Patch, OCP)는 패치 필터링, 패치 순위화, 패치 조합의 과정으로 이루어진 패치 최적화 단계[7]를 거친다. 최적화된 패치들은 패치 검증을 통해 문법적으로 오류가 없는지 (compilable), 문법적 오류도 없고 테스트케이스도 모두 통과하는지 (plausible) 확인하고, 테스트케이스까지 모두 통과한 패치들은 개발자가 직접 확인하여 올바르게 수정된 패치(correct)인지 검증한다.

[1] 본 연구성과물은 2024년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. RS-2024-00465887)

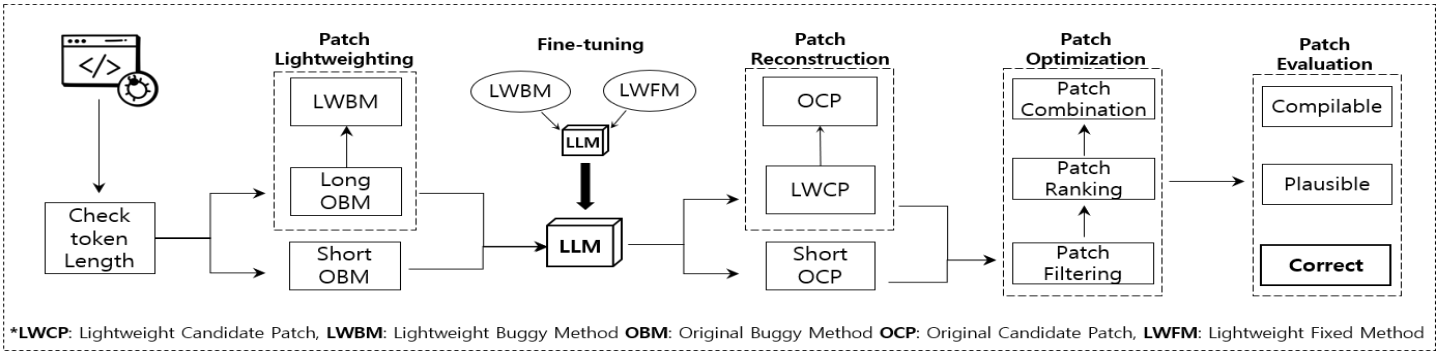

그림 1. 전체 개요

3.2. 패치 경량화

패치 경량화는 버그 메서드에서 버그가 발생한 위치를 기준으로 거리 점수와 공통식별자 점수를 종합해서 구한 최종 점수를 통해, 낮은 점수를 받은 라인들을 제거하여 LLM이 처리할 수 있는 토큰 개수 이하가 되도록 메서드를 구성하는 방법이다. 패치 경량화가 진행되는 과정은 Algorithm1과 같다. 먼저 버기 메서드의 토큰 개수를 확인하여 LLM이 처리 가능한 토큰 개수 이하인 짧은 메서드는 경량화를 진행하지 않는다. 만약 토큰 개수가 많은 긴 메서드는 경량화 과정을 거친다. 경량화는 식별자 점수와 거리 점수를 최종 종합하여 최대 점수가 1이 되는 값을 통해 진행된다. 식별자 점수는 버그 발생한 라인과 다른 라인들의 공통된 식별자 개수 비율을 통해 구하고, 거리 점수는 버그가 발생한 라인과 다른 라인들에 부여된 라인 번호의 차에 1을 더한 후 역수를 취한 값을 통해 구한다.[8] 이렇게 구한 값을 통해 총 토큰의 개수가 LLM이 처리 가능한 토큰 개수 이하가 될 때까지 낮은 점수를 가진 라인들을 제거한다. 이 과정을 통해 버그가 있는 라인과 공통된 식별자를 가지거나 가까운 위치에 있는 라인들이 버그 라인과 함께 경량화된 메서드를 구성한다. 따라서 비록 경량화된 메서드이지만, 한정된 토큰 개수 안에서 LLM이 더 많은 버그를 수정할 수 있도록 중요한 정보를 효과적으로 제공한다.

3.3. 패치 복원

패치 복원은 버그 토큰이 있는 경량화된 버기 메서드(LWBM)와 원래의 버기 메서드(OBM) 그리고 경량화된 후보패치(LWCP)와의 공통적인 특성을 활용하여, 경량화된 후보패치(LWCP)를 최종으로 온전한 메서드 형태를 가지는 후보패치(OCP)로 만들어주는 방법이다. 각 후보패치 및 메서드는 표1과 같은 특성을 가진다. 먼저 LWCP와 OCP는 수정된 코드를 가지고 있지만 버그 토큰은 없다. LWBM과 OBM은 수정된 코드는 없지만 버그 토큰은 가지고 있다. 따라서 두 버기 메서드(LWBM, OBM)와 후보패치(LWCP, OCP)의 양쪽 특성, 즉 버그 토큰과 수정된 코드를 모두 가지고 있는 LWCP+를 만들어서 최종적으로 LWCP가 OCP로 복원되어 검증단계를 거칠 수 있도록 한다.

LWCP와 OCP는 모두 수정된 패치의 내용을 가지고 있지만, 이 둘은 동일한 내용 구조를 가지고 있지 않기 때문에 버그 토큰이 있는 LWBM, OBM과의 공통점을 이용하여 LWCP가 OCP로 복원될 수 있도록 한다. 먼저 경량화된 버기 메서드(LWBM)와 경량화된 후보패치(LWCP)는 모두 선별된 statement 구조를 가지고 있다는 공통점을 이용하여, 모델이 수정한 statement부분을 버그 토큰을 가진 경량화된 버기 메서드(LWBM)의 버그 토큰 위치에 대체하여 버그 토큰을 가진 경량화된 후보패치(LWCP+)를 생성한다. 실제 생성된 LWCP+는 그림 2 (b)를 통해 확인할 수 있다. LWCP+는 그림 2 (a)의 LWBM와 동일하게 경량화된 메서드 형태를 가지고 있다.

Algorithm 1: Lightweight Process

Input: BM: Buggy Method
Output: LWBM: Lightweight Buggy Method

```
tokens ← calculate_tokens(BM)
if token_count(tokens) < LLM_token_length then
    LWBM ← BM
else
    identifiers ← extract_identifiers_from_lines(BM)
    identifier_scores ← calculate_identifier_score(identifiers)
    distance scores ← calculate_distance_score(BM)
    calculate_total_scores(BM, identifiers_scores, distance scores)
    while token_count(BM) > LLM_token_length do
        remove_lowest_score_line(BM)
        tokens ← calculate_tokens(BM)
    end while
    LWBM ← BM
end if
Return LWBM
```

표1. 패치 및 메서드 특징

	LWCP	LWBM	LWCP+	OBM	OCP
Bug Token	X	O	O	O	X
Fixed Code	O	X	O	X	O

No.	Java Code
4	externsRoot.detachChildren();
…	…
32	boolean stalenputs = false;
33	<bug> if (options.dependencyOptions. needsManagement() && options.closurePass) { </bug>
34	for (CompilerInput input : inputs) {
…	…
63	if (devMode) {
	Total: 60 lines, 512 tokens

(a) Lightweight Buggy Method(LWBM)

No.	Java Code
4	externsRoot.detachChildren();
…	…
32	boolean stalenputs = false;
33	<bug> if (options.dependencyOptions. needsManagement()) { </bug>
34	for (CompilerInput input : inputs) {
…	…
63	if (devMode) {
	Total: 60 lines, 507 tokens

(b) Lightweight Candidate Patch+(LWCP+)

No.	Java Code
1	Node parseInputs()
…	…
32	boolean stalenputs = false;
33	if (options.dependencyOptions. needsManagement()) {
34	for (CompilerInput input : inputs) {
…	…
82	}
	Total: 82 lines, 742 tokens

(c) Original Candidate Patch(OCP)

그림 2. 패치 복원 과정(Closure18)

하지만 LWBM에는 올바르게 수정된 패치의 내용이 없기 때문에 line 33에 LWCP 에서 올바르게 수정된 내용을 대체하여 그림 2 (b)처럼 LWCP +을 생성한다. LWCP +는 수정된 내용을 포함하지만 버그 토큰도 있다는 특징을 가지게 되고, 이를 다시 버그 토큰 있지만 올바르게 수정된 내용은 없는 OBM과 버그 토큰 사이의 내용을 대체한 후, 버그 토큰을 제거하여 그림 2 (c)처럼 최종 OCP를 생성한다. OCP는 OBM처럼 온전한 형태를 가지는 동시에 수정된 패치 내용도 가지게 되며, 이렇게 최종 복원된 메서드(OCP)는 패치 최적화 단계를 진행한다.

4. 실험

본 연구에서는 토큰 길이가 512개까지 처리 가능한 CodeBERT 모델을 사용하였다. CodeBERT가 경량화된 메서드 구조를 이해할 수 있도록 Bugs2fix[9] 데이터셋 중 메서드 길이가 긴 메서드들은 경량화된 형태로 변형하여 Fine-tuning하였다. 실제 검증은 Defects4j[10] 1.2버전 데이터에 실제 버그 위치를 제공하여 실험을 진행하였다. 메서드당 각 500개의 패치를 생성하였고, 사용한 Defects4j와 Bugs2fix 데이터 모두 자바코드 데이터이다.

실험은 NVIDIA RTX A6000 그래픽카드 1개, Intel Xeon Gold 6226R 2.9GHz CPU 2개, 512GB RAM, Ubuntu 22.04 LTS 환경에서 진행되었다.

경량화된 버기 메서드를 투입하여 Fine-tuning한 CodeBERT가 후보패치가 생성하면, 패치 복원 방법을 통해 경량화된 메서드를 온전한 메서드로 만든 후 패치 최적화를 거친 후 검증을 통해 올바르게 버그를 수정했는지 확인하였다. 실험을 통해 MCRepair에서 토큰 제한 문제로 올바르게 수정하지 못한 77개의 버그 중 본 방법을 통해 7개의 버그를 올바르게 수정하였다. 수정한 7개 버그 정보는 표2와 같다. 수정한 메서드들의 평균 토큰 수는 869개이고, 최대 1,975개의 토큰을 가지고 있었지만, LLM은 경량화된 정보만으로 올바르게 버그가 발생한 위치에 버그를 수정했고, 복원 과정을 거친 수정된 메서드는 테스트케이스도 모두 통과하고, 최종적으로 올바르게 버그를 수정한 것을 확인했다. 특히 Chart7, Math38, Math50은 연속하지 않은 위치의 여러 줄에 거친 멀티 청크 버그임에도 수정이 필요한 위치의 버그를 모두 올바르게 정정했다. 또한 버그를 수정한 패턴을 분석한 결과, Chart7, Closure31, Closure73, Lang24, Math38은 버그가 발생한 부분을 다른 토큰들로 대체(Replace)하여 버그를 수정했고, Closure18과 Math50은 해당 위치의 토큰들을 제거(Delete)하여 버그를 수정했다.

패치 경량화와 패치 복원 방법을 적용하였는데도 수정하지 못한 토큰 개수가 많은 버기 메서드를 분석한 결과, 다수의 버기 메서드는 한 곳 또는 그 이상의 위치에 새로운 코드의 삽입(Insert)해야 수정할 수 있는 버그임을 확인하였다. 따라서 CodeBERT모델보다 생성에 강점을 발휘하는 LLM을 사용한다면 더 많은 버그를 수정할 것으로 기대된다. 또한 단순한 버그이지만 수정을 못한 경우를 살펴보면, 먼저 경량화된 메서드 내에 대체되어야 하는 적절한 메서드 명이 포함되지 않는 경우 버그를 올바르게 수정하지 못하였다. 이를 위해서는 버그가 발생한 메서드 범위 외 추가적인 정보를 함께 제공한다면 더 많은 버그를 수정할 것으로 기대된다. 또한 일부 멀티 메서드 버그(Multi-Method Bug)의 경우 LLM이 토큰 개수가 많은 버기 메서드는 올바르게 버그를 수정했지만, 토큰 개수가 적은 메서드에서 발생한 버그는 수정하지 못해 최종적으로 전체 프로젝트는 테스트 케이스를 통과하지 못한 경우도 있었다. 이러한 부분을 통해 길이가 짧은 버그 메서드도 긴 메서드와 마찬가지로 더 많은 버그를 수정할 수 있도록 추가적인 방안을 함께 고려해야 Defects4j에서 더 많은 프로젝트를 수정할 수 있음을 의미한다.

표 2. 수정한 버그 정보

	Multi Chunk	Fix Pattern	Token Length
Chart7	O	Replace	578
Closure18	X	Delete	742
Closure31	X	Replace	903
Closure73	X	Replace	582
Lang24	X	Replace	657
Math38	O	Replace	1,975
Math50	O	Delete	649

5. 결 론

본 연구를 통해 버그가 발생한 방대한 메서드를 수정할 때 사용하고자 하는 LLM의 토큰 개수 제한이 있다면 버그를 수정할 때 필요한 정보만을 추출하여 모델에 투입하여도 버그를 정정하는 것이 가능한 것을 확인하였다. 실험을 통해 MCRepair에서 LLM의 토큰 길이 제한으로 인해 수정하지 못한 7개의 버그를 패치 경량화와 패치 복원 방법을 통해 추가로 정정하였다. 더 많은 코드 길이가 긴 버기 메서드를 수정하기 위해서는 코드 삽입에 강점을 갖는 LLM을 사용하고, 단순히 버그가 발생한 메서드 외 추가적인 정보를 함께 제공한다면 더 많은 버그를 정정할 것으로 기대된다. 또한 향후 연구에서는 메서드를 무리하게 분리하지 않고, 효과적으로 토큰 개수를 처리할 수 있는 범위로 조정하여 더 많은 토큰 길이 제한을 초과한 메서드를 수정하는 연구를 진행할 계획이다.

6. 참고 문헌

[1] J. Kim and B. Lee, “MCRepair: Multi-Chunk Program Repair via Patch Optimization with Buggy Block,” in Proc. of the 38th ACM SIGAPP Symposium on Applied Computing (SAC), 2023, pp. 1508–1515.

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

[3] A. Silva, S. Fang and M. Monperrus, "RepairLLaMA: Efficient Representations and Fine-Tuned Adapters for Program Repair," arXiv preprint arXiv:2312.12950, 2023.

[4] B. Roziere et al, "Code Llama: Open Foundation Models for Code", arXiv preprint arXiv:2308.15698, 2023.

[5] 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), 2024, pp. 1-13.

[6] C. Raffel et al, "Exploring the Limits of Transfer Learning with Unified Text-to-Text Transformer", Journal of Machine Learning Research, 2020, pp. 1-67.

[7] A. Safarovich, J. Kim, and B. Lee, “Improving Patch Optimization for Multi-Chunk Bugs in Automated Program Repair,” in Proc. of the Korea Computer Congress, 2023, pp. 200-202.

[8] E. Jung, A. Safarovich and B. Lee, “Towards Effectively Resolving Token Length Limits of LLM Models for Automatic Program Repair,” in Proc. of the Korea Computer Congress, 2024, pp. 360-362.

[9] 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), 2019, pp. 1-67.

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