

Aligning the Objective of LLM-based Program Repair

Junjielong Xu and Ying Fu and Shin Hwei Tan and Pinjia He

ICSE'25

概要

タイトル

- LLMに基づくプログラム修復の目的の整合

現状

- 自動プログラム修復(ARP)はまずまずの精度を発揮しており、特にデコーダのみのLLM（GPTシリーズ等）がよく用いられている.
- 文レベルの欠陥箇所特定（FL）ツールを用いて潜在的なバグのあるハンクの順位付きリストを取得し、次にAPR手法を用いてこれらのハンクでバグのあるプログラムを修正してパッチを生成.

概要

| 課題

1. LLMの利用方法と学習方法が異なっている.

- デコーダのみのLLMは次のトークンを予想する形式で学習されているが, ARPにおいてはインフィリング形式で使用されている.

2. ワークフローがLLMの活用を制限している.

- LLMは以前よりも進歩しており, コードとそれ以外の要素を活用し, バグを特定する能力を持っている.

概要

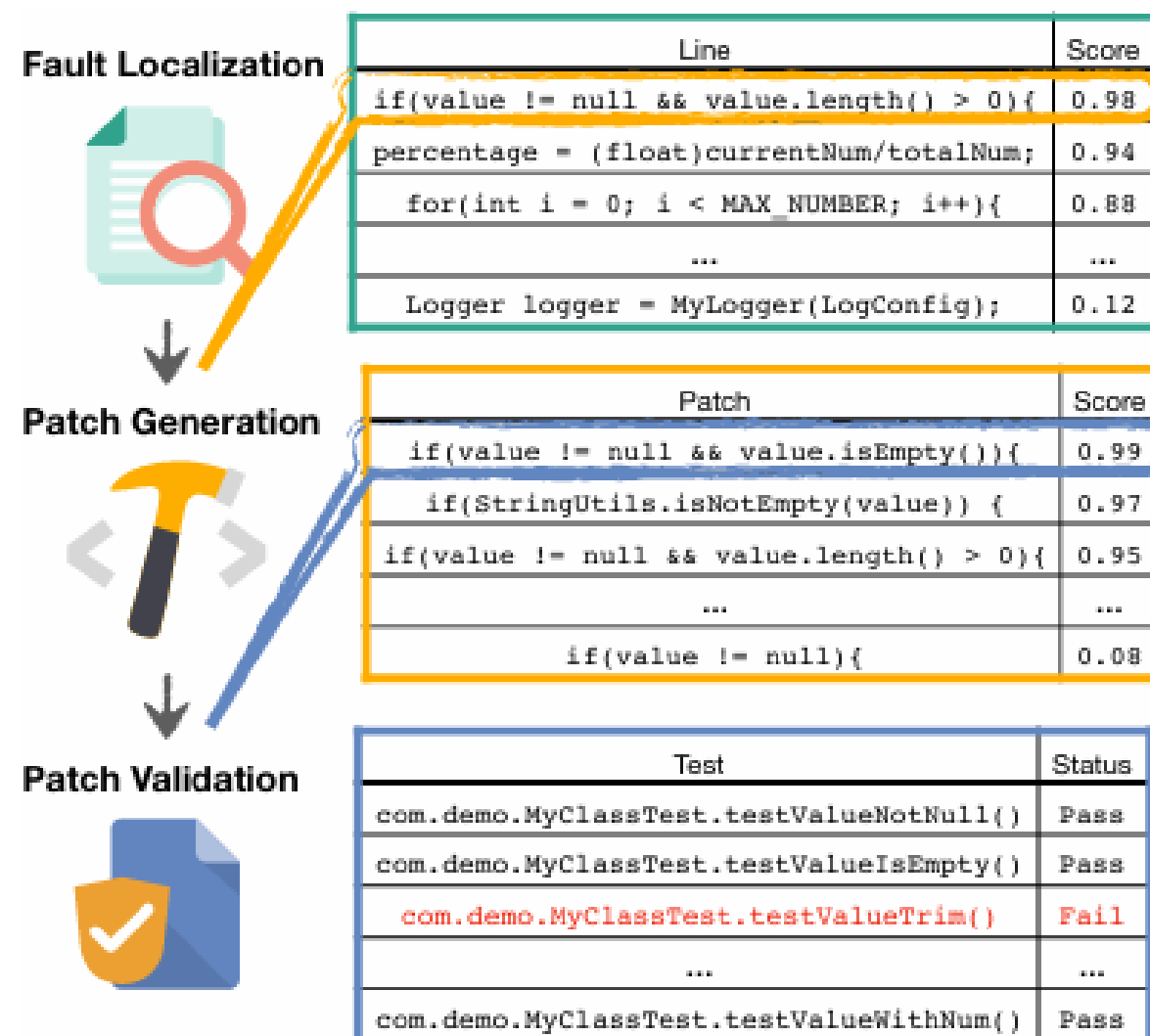


Fig. 2. An example of automated program repair workflow.

概要

目的

- デコーダのみの最新のLLMの性能を最大限生かすことのできるARP手法を開発する.

改善点

1. LLMの出力を修正されたハンクから洗練されたプログラム全体に変更することで, 推論目的を訓練目的により良く整合させる.
2. LLMにバグのあるプログラムと対応するアーティファクトをプロンプトとして与えることで, 文レベルのFLなしでバグのあるハンクを同時に特定し修復することを可能にする.

アプローチ

DIRECT DEBUG DRIVES DECENT CODE(D4C)

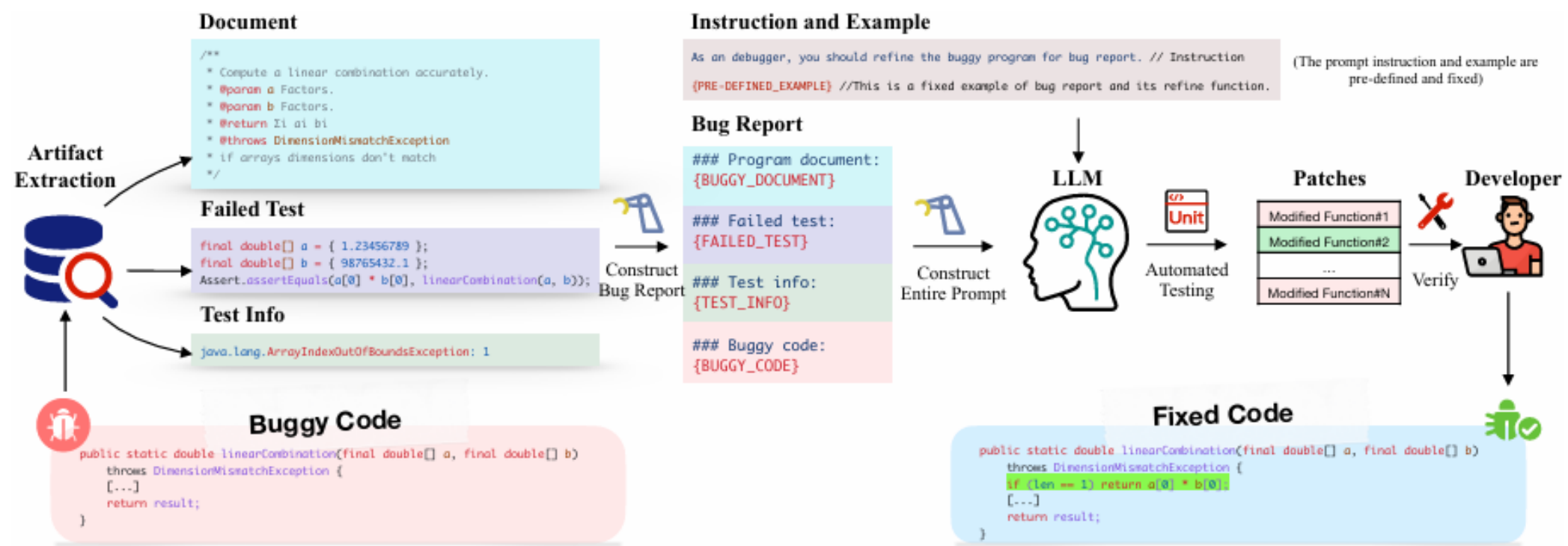


Fig. 4. The workflow of D4C. It uses the buggy code and its corresponding documents, failed tests, and test info (e.g., error message) to construct the prompt for one-shot prompting-based program repair without a specific buggy hunk (usually provided by statement-level FL tools).

アプローチ

1. アーティファクトを抽出する

- バグの関連情報（ドキュメント，失敗したテスト，エラーメッセージ）を集める.

2. プロンプト作成

- 役割を与えたうえで，一つの例とアーティファクトを与える.

3. パッチ生成

- 修正候補を10個程度生成する.

4. テスト

- 生成したコードに置き換えて一通りのテストを実行する.

評価（実験設定）

モデル

- GPT-4（ブラックボックス）, Mixtral-MoE（ホワイトボックス）

データセット

- Defects4JとDebugBenchから合計1027件の単一関数に存在するバグ

メトリクス

- 意味的にもバグを修正できた場合のみを成功とする.

比較相手

- AlphaRepair, ChatRepairなどの既存の最新の手法

評価

RQ1 D4Cは既存の手法より優れているか？

条件 1 完璧な環境での性能（バグの場所を把握している）

- 180件の再現に成功（従来手法から10%の向上）

TABLE II
THE NUMBER OF CORRECT PATCHES GENERATED BY DIFFERENT
APPROACHES ON DEFECTS4J v1.2 & v2.0.

Method	Model	v1.2	v2.0	Sum	Sampled
AlphaRepair	CodeBERT	52	34	86	5000
Repilot	InCoder	66	50	116	5000
RAP-Gen	CodeT5	72	53	125	100
FitRepair	CodeT5	89	44	133	5000
ChatRepair	GPT-4	114	48	162	100-200
D4C	GPT-4	84	96	180	10

評価

RQ1 D4Cは既存の手法より優れているか？

条件 2 現実的な環境での性能

- 従来手法が30%以上精度が下がったのに比べ、D4Cは4.8%の低下に留めた.

TABLE III
THE NUMBER OF CORRECT PATCHES GENERATED IN PERFECT FL AND
STATISTICAL FL ON DEFECTS4J v1.2.

Method	Model	Perf.	Stat.	Drop	Sampled
AlphaRepair	CodeBERT	52	36	30.8%	5000
RAP-Gen	CodeT5	72	48	33.3%	100
D4C	GPT-4	84	80	4.8%	10

評価

RQ2 D4Cを支える2つのアイデアは本当に効果的か？

アイデア 1 「関数全体」を出力させる方が良い

- ハンクだけを出力させる場合よりも，関数で出力させた場合の方がパープレキシティが低下し，精度も向上した。

TABLE IV
COMPARISON OF DIFFERENT SETTINGS OF D4C ON DEBUGBENCH. WE MEASURE PERPLEXITY AND VERIFIED PATCHES TO VALIDATE INSIGHTS.

Format	Perplexity		Mixtral (#Verified)			GPT-4 (#Verified)		
	O	IO	C++	Java	Python	C++	Java	Python
Mask-Hunk	8.59	2.68	66	39	63	140	139	129
Mask-Func	3.01	1.58	99	91	76	160	153	147
Report-Hunk	8.50	2.64	72	64	67	144	140	133
Report-Func	1.39	1.79	118	125	104	177	163	161

評価

RQ2 D4Cを支える2つのアイデアは本当に効果的か？

アイデア2 「バグレポート」を入力する方が良い

- 従来のアプローチであるバグ箇所をマスクする手法と比べて、バグレポートを与えた場合の方がパープレキシティが低下し、精度も向上した。

TABLE IV
COMPARISON OF DIFFERENT SETTINGS OF D4C ON DEBUGBENCH. WE MEASURE PERPLEXITY AND VERIFIED PATCHES TO VALIDATE INSIGHTS.

Format	Perplexity		Mixtral (#Verified)			GPT-4 (#Verified)		
	O	IO	C++	Java	Python	C++	Java	Python
Mask-Hunk	8.59	2.68	66	39	63	140	139	129
Mask-Func	3.01	1.58	99	91	76	160	153	147
Report-Hunk	8.50	2.64	72	64	67	144	140	133
Report-Func	1.39	1.79	118	125	104	177	163	161

評価

RQ3 D4Cの各要素はどれくらい重要か？

- プロンプトに与える要素の効果について，アブレーション研究を用いて検証.

TABLE V
THE NUMBER OF PATCHES GENERATED FOR OUR ABLATION STUDY AND SENSITIVE ANALYSIS. WE MEASURE PLAUSIBLE PATCHES FOR DEFECTS4J AND PATCHES VERIFIED VIA UNSEEN TESTS FOR DEBUGBENCH.

Benchmark	Defects4J (#Plausible)			DebugBench (#Verified)			
	v1.2	v2.0	Sum	C++	Java	Python	Sum
w/o Document	93	99	192	175	162	158	495
w/o Test	80	89	169	173	159	159	491
w/o Message	81	93	174	169	162	161	492
- Mask	53	73	126	160	153	147	460
- Pure	47	62	109	143	140	131	414
Samp=1	60	73	133	163	151	145	459
Samp=3	91	88	179	177	163	161	501
Samp=10 (default)	96	114	210	181	171	169	521
Temp=0.0	61	75	136	165	154	146	465
Temp=1.0 (default)	96	114	210	181	171	169	521

結論と所感

結論

- D4Cは使用方法と学習方法を整合することと、不要な手順（FL）を排除することによって、APRの精度を向上させた.

所感

- ただ良いものを使うだけでなく、原理からより良い使い方を考えるという発想が良かった.

TABLE I
STATISTICS OF THE BENCHMARKS: THE NUMBER OF SINGLE-FUNCTION
BUGS AND THE THREAT OF DATA LEAKAGE

Benchmark	Bug number	Data leakage	Language
Defects4J	437	Yes	Java
DebugBench	200/194/196	No	C++/Java/Python3