An Empirical Comparison of LLMs on C/C++ Program Repair

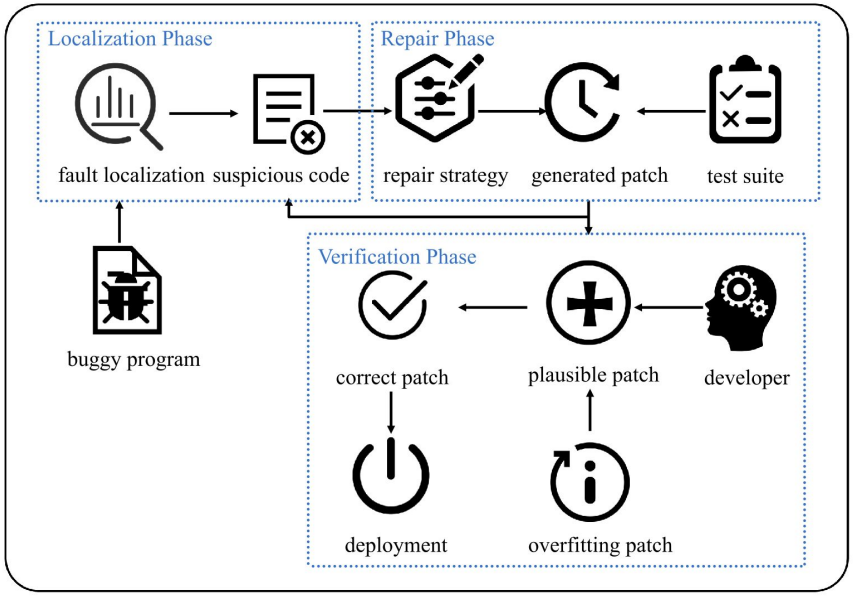
Team 3 Background Presentation

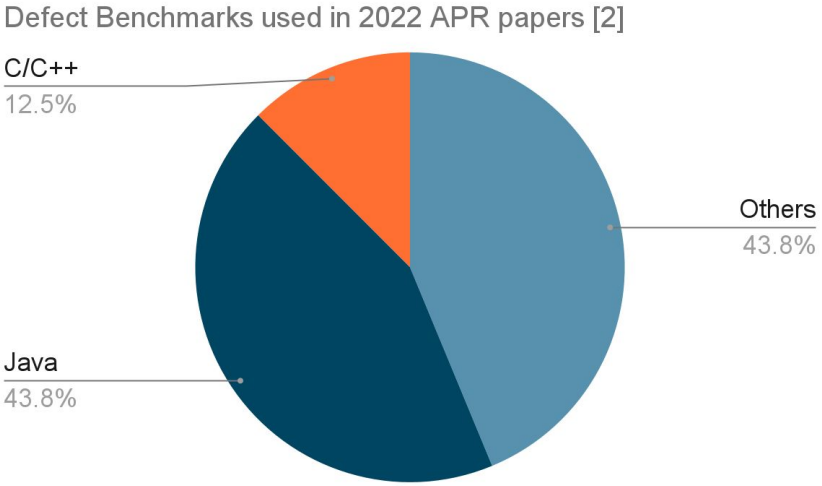
Xinzhuo Hu, Musheng He, Rishika Garg
Srikkanth Ramachandran, Harshil Bhandari

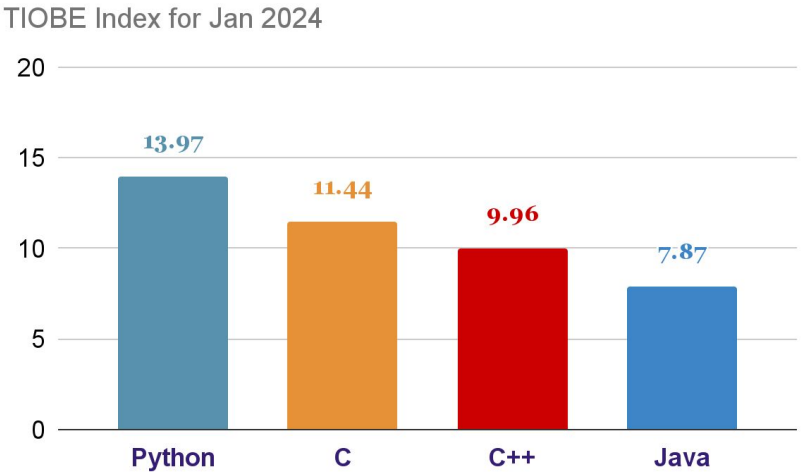
2024-01-30

Background

Overview of Automated Program Repair(APR) [1]





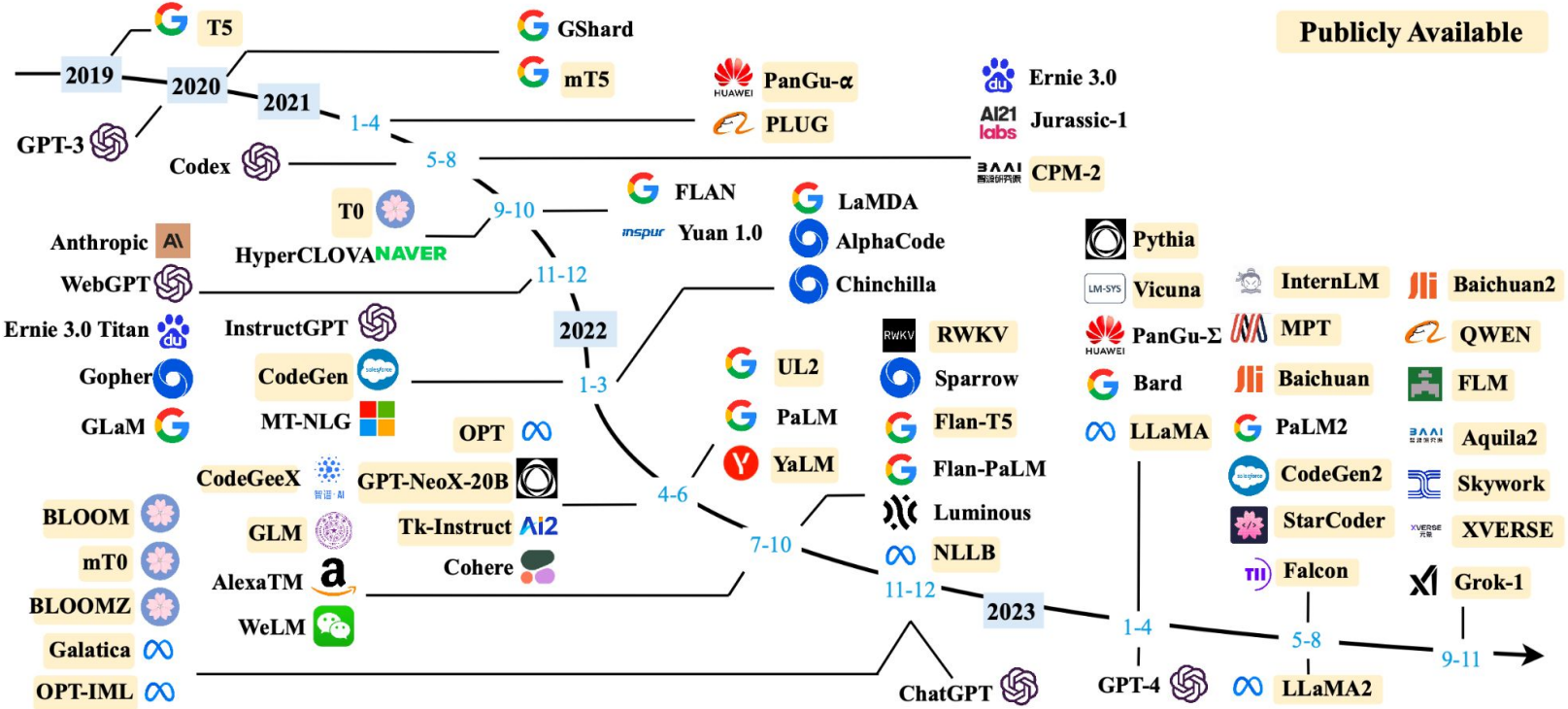


[1] Q. Zhang, C. Fang, Y. Ma, W. Sun, and Z. Chen, “A Survey of Learning-based Automated Program Repair,” *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 2, p. 55:1-55:69, Dec. 2023, doi: 10.1145/3631974.

[2] G. An, M. Kwon, K. Choi, J. Yi, and S. Yoo, “BUGSC++: A Highly Usable Real World Defect Benchmark for C/C++,” *2023 38th IEEE/ACM ASE*, IEEE Computer Society, Sep. 2023, pp. 2034–2037. doi: 10.1109/ASE56229.2023.00208.

Background

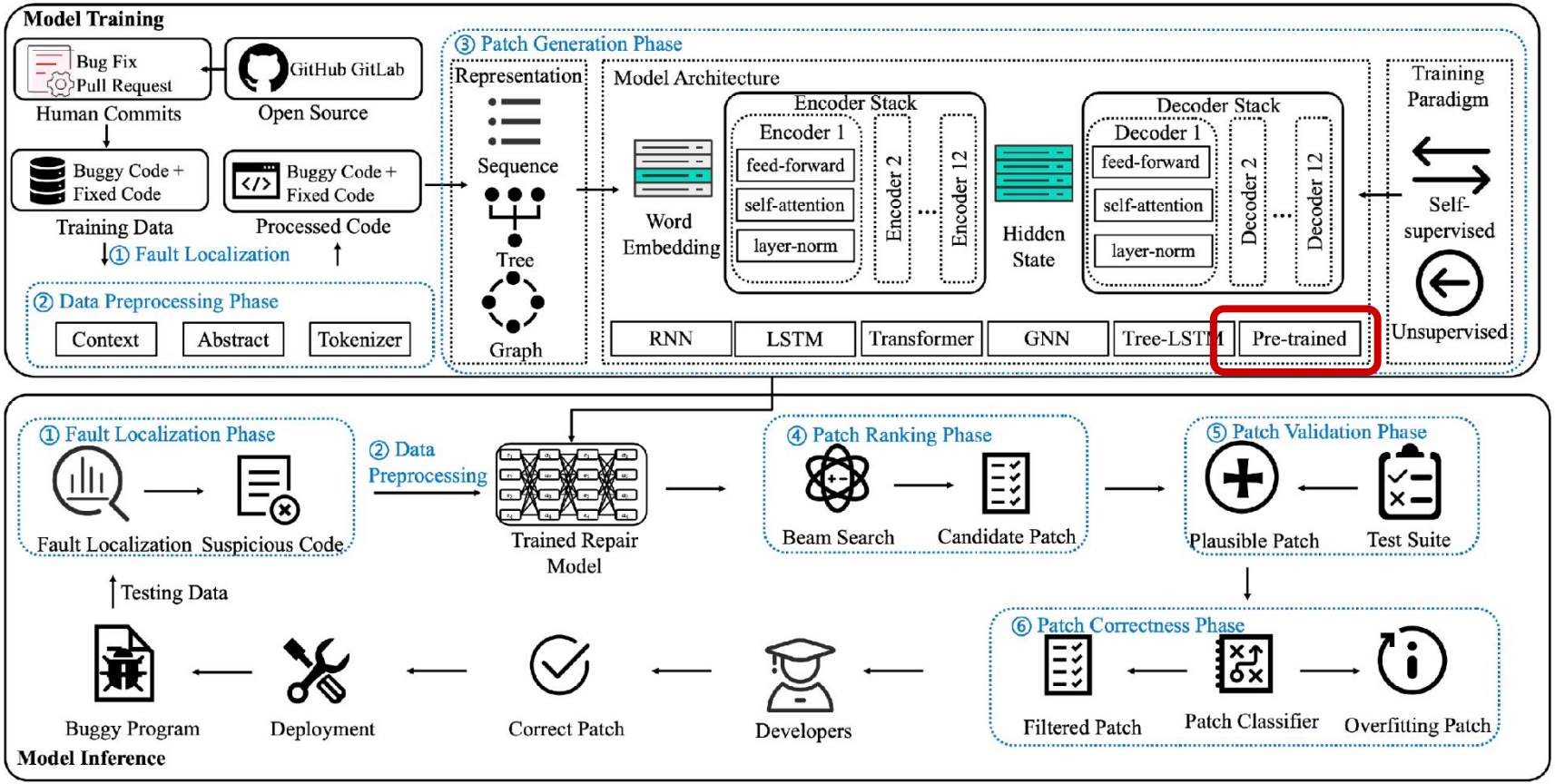
Large Language Model (LLM) [3]



[3] W. X. Zhao *et al.*, "A Survey of Large Language Models." arXiv, Nov. 24, 2023. doi: 10.48550/arXiv.2303.18223.

Background

Detailed Workflow of Learning-based APR [1]



[1] Q. Zhang, C. Fang, Y. Ma, W. Sun, and Z. Chen, "A Survey of Learning-based Automated Program Repair," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 2, p. 55:1-55:69, Dec. 2023, doi: 10.1145/3631974.

Problem

Empirical Studies are mostly on pre-trained LLMs

Limited language benchmarks

- Java

Limited bug types

- Single-hunk
- Vulnerabilities for C/C++

A Summary and Comparison of Empirical Studies in LLM-based APR

Year	Study	Language	Description
2021	Ehsan Mashhadi and Hadi Hemmati. Applying codebert for automated program repair of Java simple bugs. In MSR'21. 505–509.	Java	CodeBERT in fixing bugs from the ManySStuBs4J benchmark
2022	Kolak et al. Patch generation with language models: Feasibility and scaling behavior. In ICLR-DL4C'22.	Java	Investigates the performance of the pre-trained model in fixing bugs from the QuixBugs benchmark
2022	Xia et al. Automated Program Repair in the Era of Large Pre-trained Language Models. In ICSE'23. 1482–1494.	Java, Python, C	The first extensive study on directly applying nine pre-trained models for APR across three programming languages
2022	Kim et al. An empirical study of deep transfer learning-based program repair for kotlin projects. In ESEC/FSE'22. 1441–1452.	Kotlin	Investigates the performance of the pre-trained model in fixing defects in the Samsung Kotlin projects
2022	Huang et al. Repairing security vulnerabilities using pre-trained programming language models. In DSN-W'22. IEEE, 111-116	C/C++	A preliminary study to investigate the performance of pre-trained models in ***repairing security vulnerabilities***
2022	Fu et al. VulRepair: a T5-based automated software vulnerability repair. In ESEC/FSE'22. 935–947.	C/C++, C#	VulRepair, a T5-based automated software ***vulnerability repair approach*** that leverages the pre-training and BPE
2023	Huang et al. An Empirical Study on Fine-Tuning Large Language Models of Code for Automated Program Repair. In ASE'23. 1162-1174.	Java, JavaScript, C/C++	a comprehensive study on the repair capabilities of 5 LLMCs in various repair scenarios with pre-trained architectures

Our Goal

A comprehensive study on the program repair capabilities and generalities of **3 general-purpose LLMs** for real-world C/C++ projects

- Free-to-Use Chatbots (e.g., ChatGPT, Bard, LLaMA, Claude) are more user-friendly and accessible for general developers and students
- **BugSC++**: a new C/C++ program defect benchmark

Research Questions

RQ1: How effective are LLMs in repairing different types of bugs in C/C++ and does their effectiveness vary under different repair scenarios?

RQ2: Do LLMs perform better with additional context information in extra rounds of interactions with the user?

RQ3: How do LLMs compare with existing traditional C/C++ APR tools?

Why BugsC++?

A Highly Usable Real World Defect Benchmark for C/C++ [1]

- Easy to Use
  - similar user interface to *Defects4J*
  - Handy Command Line Interface
- **Diverse real-word bugs**
  - 209 bugs mined from 23 C/C++ open-source projects
- Reproducibility
  - Via Docker

vs.

Existing C/C++ Benchmarks:

ManyBugs
- 185 bugs in 9 open C projects
- Only 91 reproducible [2]

COREBench
- Regression errors in *2 C projects*

DeepFix, SPoC, Code4Bench…
- Scraped data from codeforces or student programming tasks

[1] G. An, M. Kwon, K. Choi, J. Yi, and S. Yoo, "BUGSC++: A Highly Usable Real World Defect Benchmark for C/C++," *2023 38th IEEE/ACM ASE*, IEEE Computer Society, Sep. 2023, pp. 2034–2037. doi: 10.1109/ASE56229.2023.00208.

[2] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated Program Repair in the Era of Large Pre-trained Language Models. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), May 2023. 1482–1494. https://doi.org/10.1109/ICSE48619.2023.00129

Papers

1. C. S. Xia, Y. Wei, and L. Zhang, “Automated Program Repair in the Era of Large Pre-trained Language Models,” In 2023 ICSE.

2. K. Huang et al., “An Empirical Study on Fine-Tuning Large Language Models of Code for Automated Program Repair,” In 2023 ASE.

3. M. Jin et al., “InferFix: End-to-End Program Repair with LLMs,” In ESEC/FSE 2023.

4. C. S. Xia and L. Zhang, “Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT.” arXiv:2304.00385, 2023

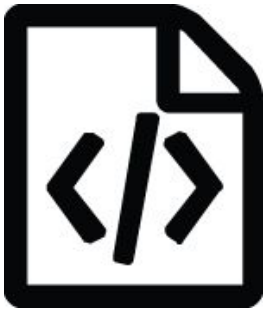
Automated Program Repair in the Era of Large Pre-trained Language Models

Chunqiu Steven Xi[1], Yuxiang Wei[1] and Lingming Zhang[1]

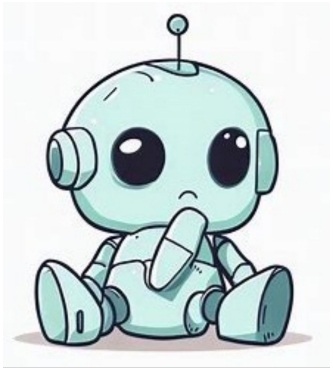
[1] University of Illinois Urbana Champaign

International Conference on Software Engineering 2023

Automated Program Repair (APR)

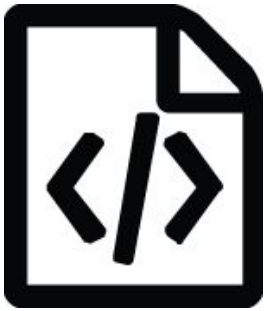


Buggy Code

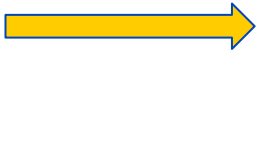


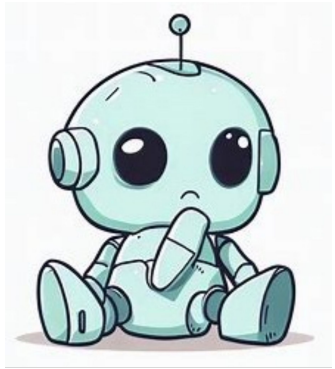
APR Tool

Automated Program Repair (APR)



Buggy Code

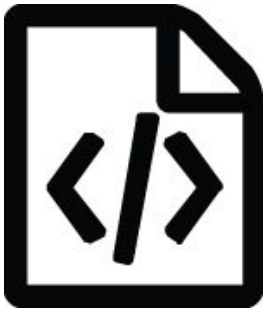




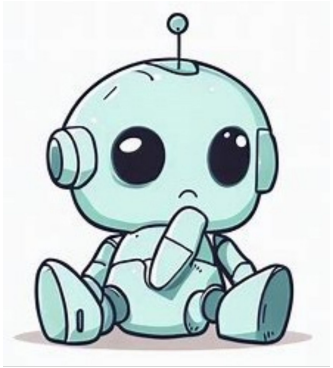
APR Tool

Suggested fix(es)

Automated Program Repair (APR)



Buggy Code

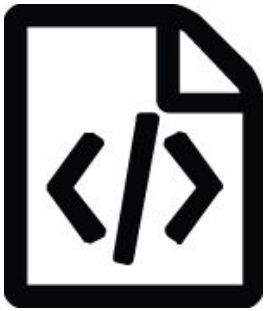


APR Tool

Suggested fix(es)

Unit Tests

Automated Program Repair (APR)



Buggy Code

APR Tool

Suggested fix(es)

plausible

fail

pass

Unit Tests

APR Tasks 1 - Function Completion

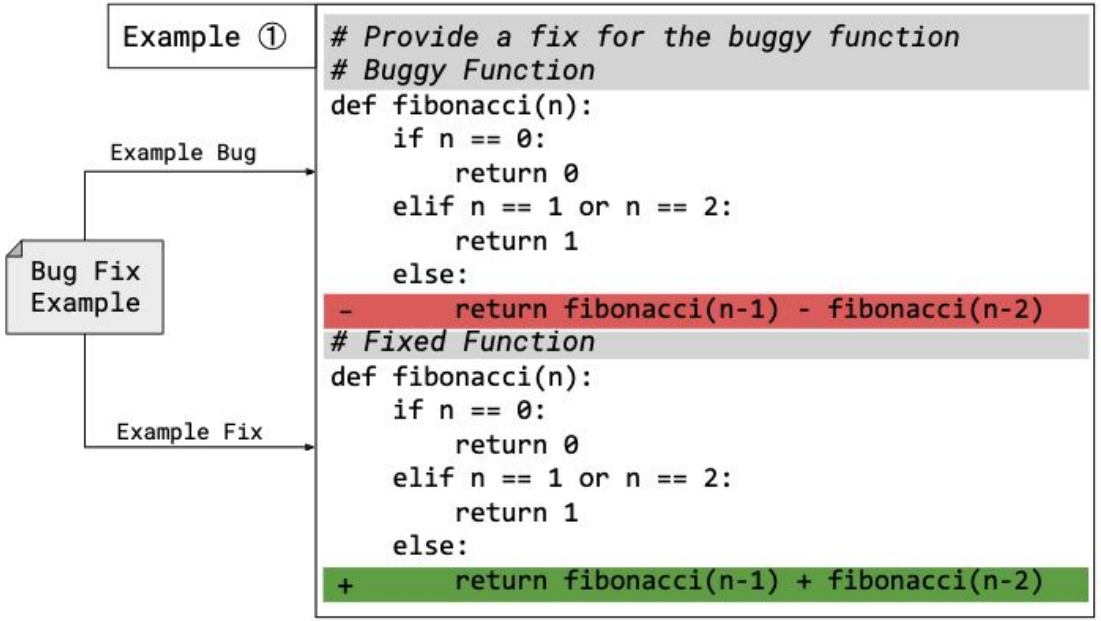


Figure taken from the paper

APR Tasks 2 - Code Infilling

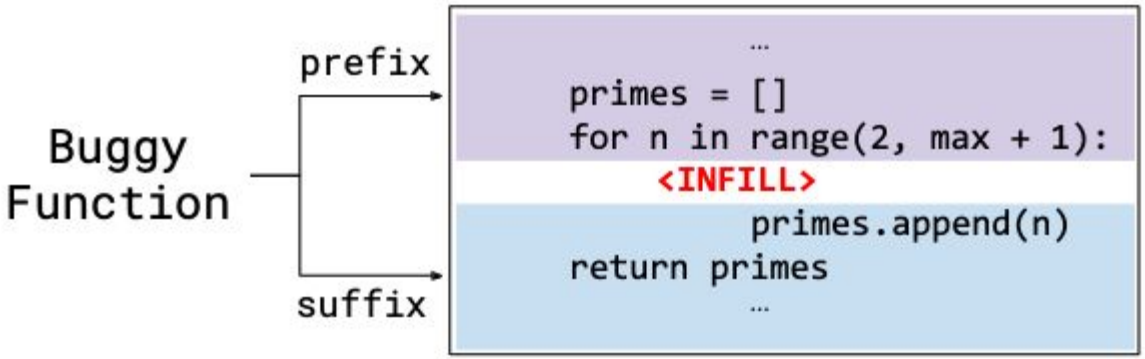


Figure taken from the paper

APR Tasks 3 - Single Line Completion

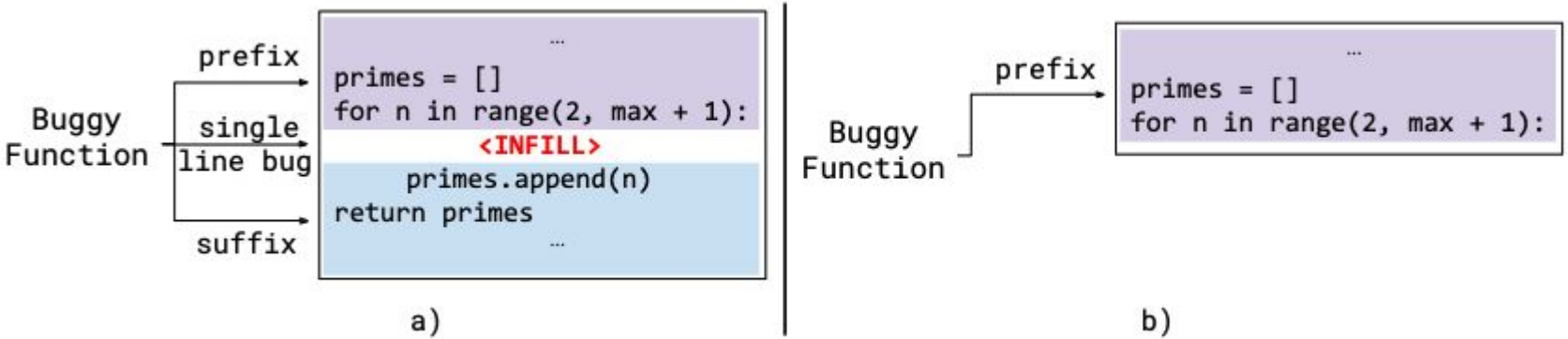
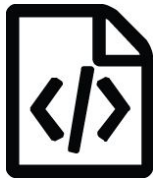


Figure taken from the paper

Methodology

Prompt with example fixes

+



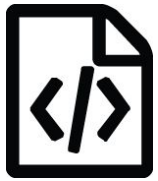
buggy code

LLM
(from Hugging Face / prompting)

Methodology

Prompt with example fixes

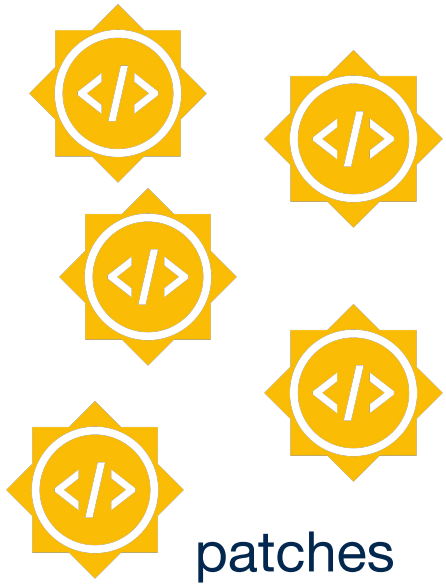
+



buggy code

LLM
(from Hugging Face / prompting)

Nucleus sampling

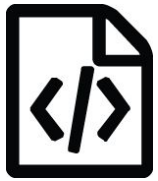


patches

Methodology

Prompt with example fixes

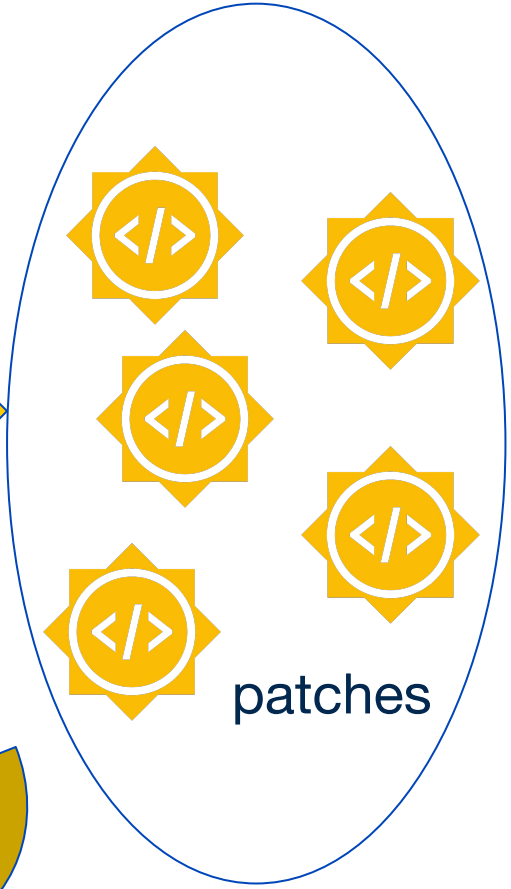
+



buggy code

LLM
(from Hugging Face / prompting)

Nucleus sampling



patches

Best candidate

Metrics

of correct patches generated

# of plausible patches generated

Execution speed of generating patches

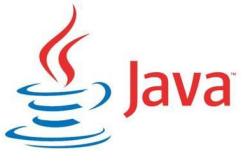
Total and mean entropy

Data Sets

rjust/**defects4j**

A Database of Real Faults and an Experimental Infrastructure to Enable Controlled Experiments in Software Engineering Research

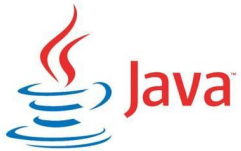
27 Contributors · 63 Issues · 628 Stars · 285 Forks



nus-apr/**quixbugs-java-benchmark**

Quixbugs Benchmark with metadata for java bugs

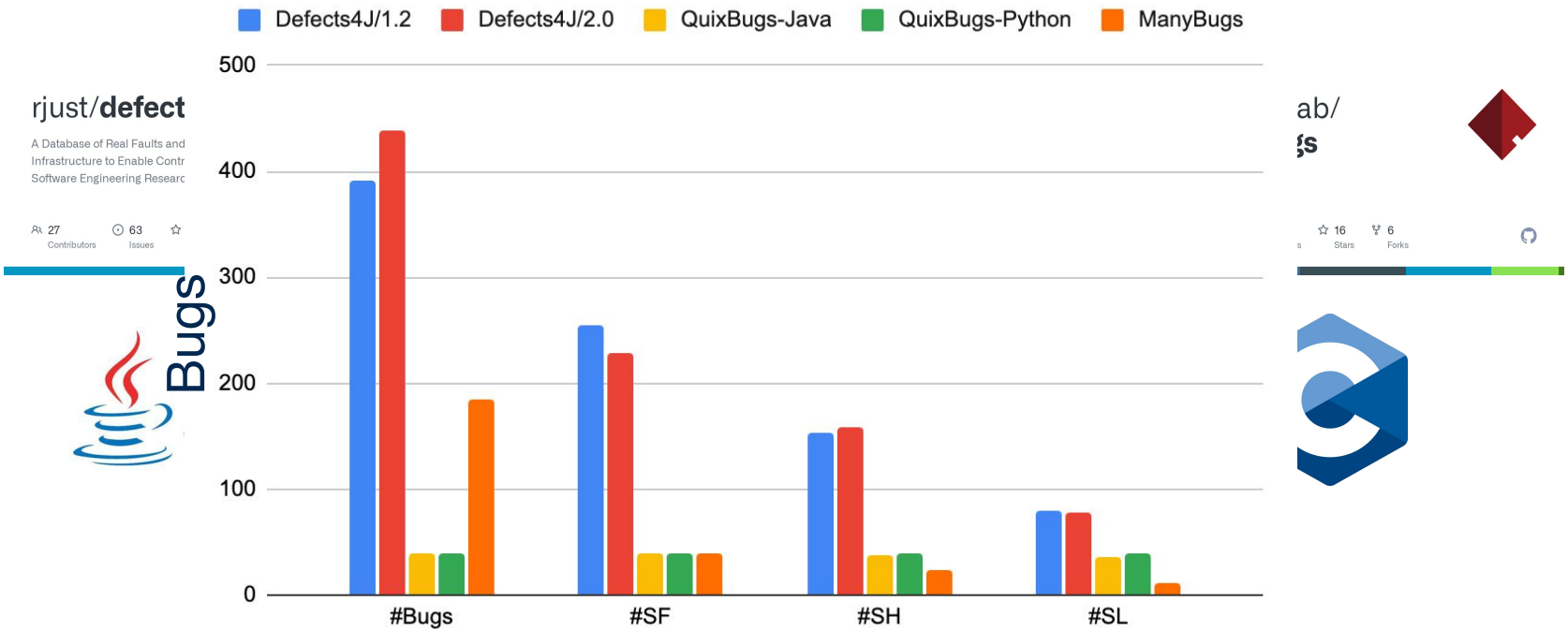
1 Contributor · 0 Issues · 0 Stars · 0 Forks



squaresLab/**ManyBugs**

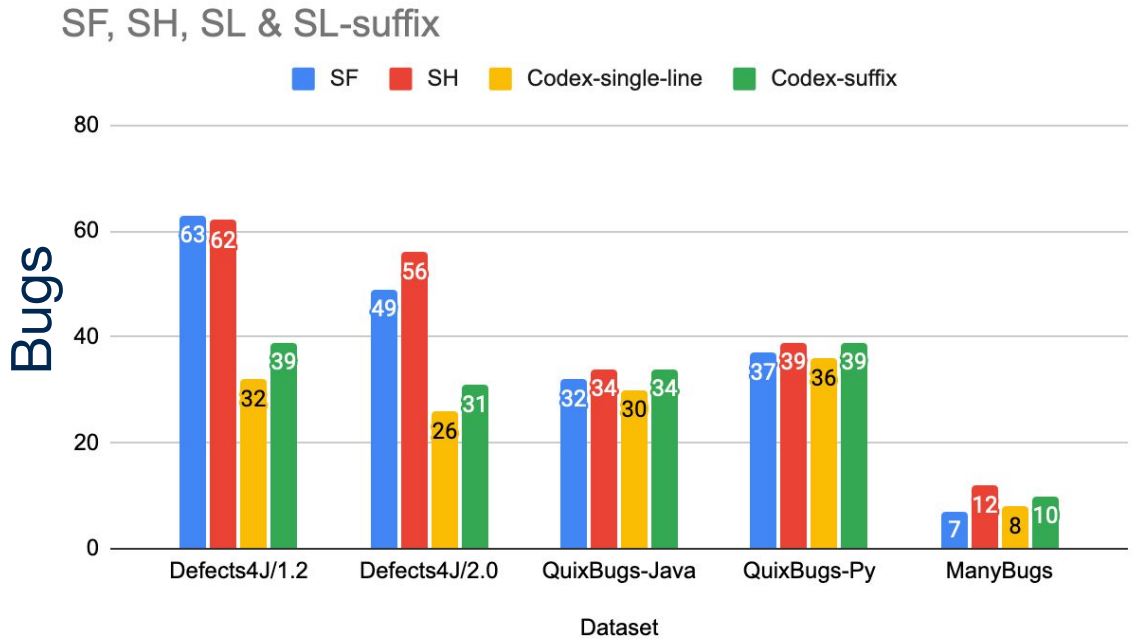
3 Contributors · 13 Issues · 16 Stars · 6 Forks

Data Sets


Defects4J/1.2
Defects4J/2.0
QuixBugs-Java
QuixBugs-Python
ManyBugs
500
400
300
200
100
0
#Bugs
#SF
#SH
#SL
rjust/defect
A Database of Real Faults and
Infrastructure to Enable Contr
Software Engineering Researc
27
Contributors
63
Issues
Bugs
ab/
gs
16
Stars
6
Forks

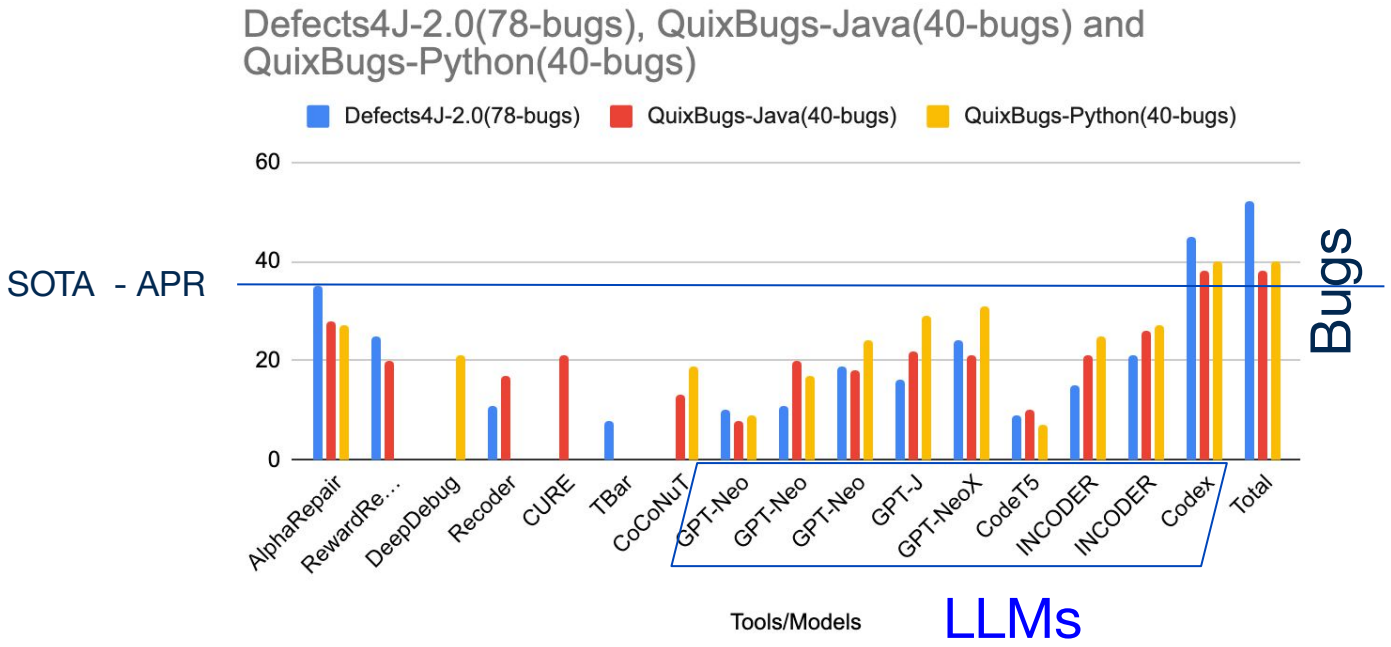
Research Questions & Results

RQ1. How effective are LLMs for the 3 APR settings?



Research Questions & Results

RQ2. How do LLMs compare with SOTA APR?



Research Questions & Results

RQ3. Can LLMs be used to rank patches?

TABLE IX: Mean entropy of generated patches

	Models	Defects4J 1.2			QuixBugs-Python		
		C	P	NP	C	P	NP
Function Gen.	GPT-Neo 125M	0.08	0.13	0.23	0.10	0.10	0.20
	GPT-Neo 1.3B	0.12	0.12	0.19	0.06	0.05	0.09
	GPT-Neo 2.7B	0.09	0.13	0.17	0.05	0.06	0.08
	GPT-J	0.07	0.10	0.12	0.04	0.05	0.08
	GPT-NeoX	0.08	0.11	0.13	0.05	0.07	0.10
	Codex	0.04	0.05	0.08	0.11	0.13	0.16
Infilling	CodeT5	0.50	0.51	0.54	0.51	0.50	0.59
	INCODER 1.3B	0.49	0.58	0.65	0.54	0.56	0.65
	INCODER 6.7B	0.45	0.50	0.61	0.61	0.60	0.65
	Codex	0.43	0.43	0.50	0.32	0.33	0.42
Line Gen.	GPT-Neo 125M	0.38	0.42	0.58	0.41	0.45	0.61
	GPT-Neo 1.3B	0.32	0.38	0.58	0.25	0.27	0.47
	GPT-Neo 2.7B	0.28	0.32	0.55	0.21	0.26	0.40
	GPT-J	0.29	0.33	0.54	0.20	0.22	0.38
	GPT-NeoX	0.39	0.42	0.71	0.26	0.28	0.55
	Codex	0.19	0.28	0.57	0.18	0.23	0.60

Res. By calculating entropy, we find Correct solutions are more “natural”.

Entropy can be used to effectively rank patches.

Research Questions & Results

RQ4. How can we improve LLM performance?

TABLE X: Further improving LLM-based APR

Tools / Models	Defects4J 1.2 All	Defects4J 2.0 Single Line	QuixBugs-Python
AlphaRepair	74	35	27
RewardRepair	50	25	-
DeepDebug	-	-	21
Recoder	65	11	-
TBar	68	8	-
INCODER (200)	37	21	27
INCODER (2000)	64	25	32
INCODER w/ template (2000)	**78**	**39**	**37**

Res. More samples
+ repair templates
=> better results

An Empirical Study on Fine-tuning Large Language Models of Code for Automated Program Repair

Kai Huang*† , Xiangxin Meng‡, Jian Zhang§¶, Yang Liu§, Wenjie Wang*, Shuhao Li†, Yuqing Zhang*†¶

*University of Chinese Academy of Sciences, Beijing, China
†Zhongguancun Laboratory, Beijing, China
‡Beihang University, Beijing, China
§Nanyang Technological University, Singapore

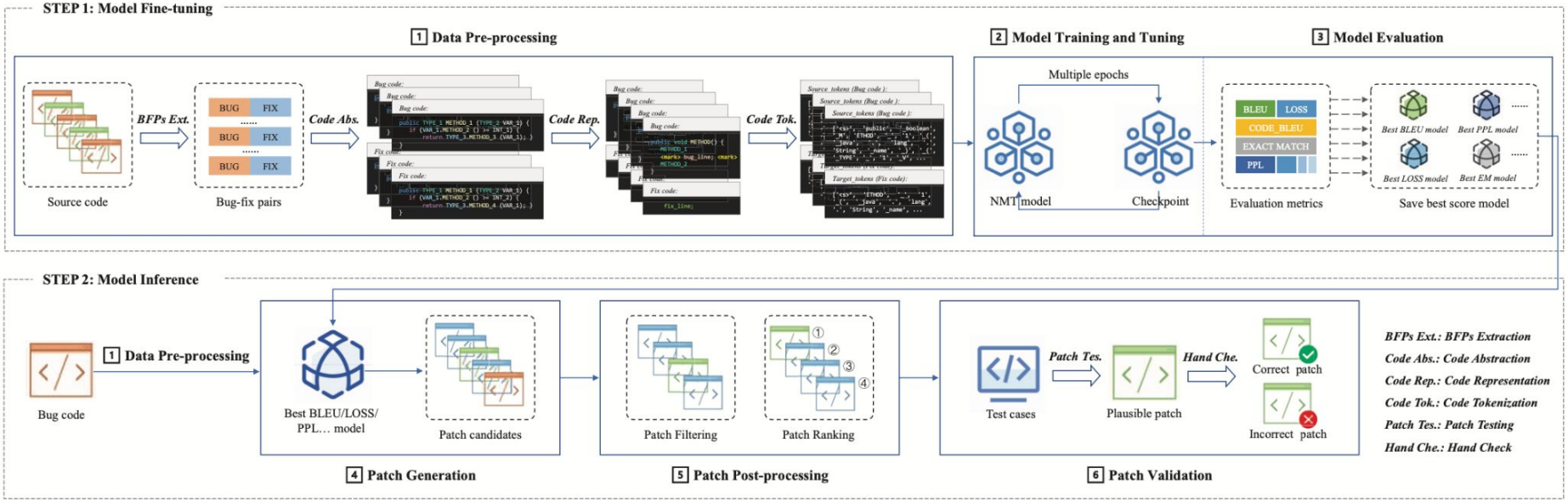
2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)

UCDAVIS

Research Questions

- **RQ1: How do different design choices affect LLMCs' repair capability?**
- **RQ2: How well does the LLMC perform compared to the state-of-the-art approaches?**
- **RQ3: What are the factors that limit the effectiveness of fine-tuning LLMCs?**

Methodology



Models

Model		CodeBERT	GraphCodeBERT	PLBART	CodeT5	UniXcoder
Model Source		Microsoft/ codebert-base [47]	Microsoft/ graphcodebert-base [48]	Uclanlp/ plbart-base [49]	Salesforce/ codet5-base [50]	Microsoft/ unixcoder-base [51]
Model Size		125M	125M	140M	220M	125M
Architecture		Encoder	Encoder	Encoder-Decoder	Encoder-Decoder	Encoder-Decoder
Pretrain Dataset		CodeSearchNet [52]	CodeSearchNet [52]	StackOverflow and BigQuery	CodeSearchNet [52] and BigQuery	CodeSearchNet [52]
Pretrain PLs	C				✓	
	C#				✓	
	C++					
	Java	✓	✓	✓	✓	✓
	Python	✓	✓	✓	✓	✓
	JavaScript	✓	✓		✓	✓
	PHP	✓	✓		✓	✓
	Ruby	✓	✓		✓	✓
	Go	✓	✓		✓	✓

Datasets and Baseline

Repair Task		Train Dataset			Test Benchmark			Language	Defect Complexity		Baseline	Parameter Setting				
Defect	Task	Dataset	#Bugs	#BFPs	Dataset	#Bugs	#BFPs		Single-hunk	Multi-hunk		I.O.	L.R.	T.E.	B.S.	P.N.
Bug	❶	BFP_small dataset	-	52,515	BFP_small dataset	-	5,835	Java	✓	✓	Tufano et al. [7]	512	5e-5	30	5	1
		BFP_medium dataset	-	58,909	BFP_medium dataset	-	6,546	Java	✓	✓	Tufano et al. [7]	512	5e-5	30	5	1
	❷	SequenceR dataset	-	35,551	SequenceR dataset	-	4,707	Java	✓		Chen et al. [10]	512	5e-5	30	50	50
	❸	Recoder dataset	-	143,666	Defects4J V1.2	383	554	Java	✓		Jiang et al. [21]	512	5e-5	30	100	10
		Recoder dataset	-	143,666	Defects4J V2.0	412	719	Java	✓		Jiang et al. [21]	512	5e-5	30	100	10
	❹	CPatMiner dataset	44,154	80,501	Defects4J V1.2	383	554	Java	✓	✓	Li et al. [23]	512	1e-4	30	100	100
Vul.	❺	VulRepair dataset	-	6,776	VulRepair dataset	-	1,706	C/C++	✓	✓	Fu et al. [17]	512	2e-5	75	50	50
Error	❻	TFix dataset	-	94,300	TFix dataset	-	10,504	JavaScript	✓		Berabi et al. [24]	512	2e-5	30	5	1

Experiment

- Task 1: Study the impact of different design choices:
  - Code Abstraction (Abs/Raw)
  - Code Presentation (CR1/CR2/CR3)
  - Checkpoints (PPL/BLEU/Last)
- Task 2: Using best combination(CR3 + Raw) on SequenceR.
- Task 3: Evaluate single-hunk bugs on Defects4J V1.2/V2.0
- Task 4: Evaluate multi-hunk bugs on Defects4J V1.2
- Task 5: Explore vulnerability repair Capability.
- Task 6: Explore Single-hunk error repair Capability.

Results

TABLE III: Repair results of LLMCs in different repair tasks. (X/Y: correct patches / plausible patches; Z%: repair accuracy)

Task	Benckmark	Code Rep.+ Abs.	CodeBERT			GraphCodeBERT			PLBART			CodeT5			UniXcoder		
			PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last
❶	BFP_S	CR1_abs	12.94%	**16.90%**	**16.90%**	13.32%	17.55%	**17.62%**	17.96%	8.86%	**19.40%**	20.15%	**21.80%**	21.37%	**19.47%**	18.89%	19.20%
		CR1_raw	12.17%	16.30%	**17.48%**	14.81%	17.26%	**17.46%**	14.41%	17.86%	**17.93%**	19.69%	22.45%	**25.18%**	18.44%	**23.79%**	**23.79%**
		CR2_raw	**34.28%**	**34.28%**	**34.28%**	31.11%	33.44%	**33.52%**	31.65%	34.14%	**34.52%**	43.27%	**47.82%**	47.80%	36.71%	41.70%	**42.33%**
		CR3_raw	34.82%	**35.25%**	34.82%	36.49%	**39.32%**	**39.32%**	33.13%	34.57%	**36.00%**	42.71%	**47.66%**	47.34%	42.67%	**45.02%**	**45.02%**

Task	Benckmark	Code Rep.+ Abs.	CodeBERT			GraphCodeBERT			PLBART			CodeT5			UniXcoder		
			PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last
❶	BFP_M	CR1_abs	4.72%	3.16%	**9.73%**	3.96%	3.96%	**10.36%**	**13.17%**	0.31%	**13.17%**	8.27%	**13.95%**	13.92%	5.39%	2.55%	**8.08%**
		CR1_raw	4.22%	**9.41%**	**9.41%**	5.27%	**10.45%**	**10.45%**	4.58%	0.69%	**10.13%**	8.72%	**16.03%**	**16.03%**	6.59%	11.38%	**11.64%**
		CR2_raw	26.11%	**31.75%**	31.61%	27.56%	30.88%	**30.96%**	23.47%	**27.96%**	**27.96%**	35.77%	**41.88%**	**41.88%**	32.21%	**36.85%**	**36.85%**
		CR3_raw	28.40%	**33.48%**	**33.48%**	29.01%	33.35%	**33.43%**	27.04%	30.39%	**30.74%**	36.24%	**42.12%**	**42.12%**	36.29%	39.53%	**39.65%**

Task	Benckmark	Code Rep.+ Abs.	CodeBERT			GraphCodeBERT			PLBART			CodeT5			UniXcoder		
			PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last
❷	SeqRD	CR3_raw	**19.06%**	14.40%	14.49%	**19.35%**	14.74%	14.53%	**17.87%**	13.13%	12.96%	**36.22%**	26.20%	26.03%	**33.91%**	22.33%	22.24%

Task	Benckmark	Code Rep.+ Abs.	CodeBERT			GraphCodeBERT			PLBART			CodeT5			UniXcoder		
			PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last
❸	D4J V1.2	CR3_raw	21/37	**25/35**	23/31	24/39	26/35	**26/37**	12/24	**16/23**	**16/23**	**41/53**	30/43	30/43	**46/63**	36/46	38/45
	D4J V2.0	CR3_raw	22/38	12/26	**23/26**	**23/43**	22/38	22/37	**23/39**	15/27	15/27	**30/39**	16/26	16/26	**37/55**	20/33	22/35
	D4J V1.2	CR3_raw	**7.76%**	6.68%	6.14%	**8.84%**	6.86%	7.04%	4.51%	**4.69%**	**4.69%**	**11.01%**	8.84%	8.84%	**11.73%**	8.48%	9.21%
	D4J V2.0	CR3_raw	**5.84%**	2.78%	3.20%	**5.70%**	4.03%	4.03%	**6.12%**	4.59%	4.59%	**8.34%**	6.12%	6.12%	**8.84%**	5.70%	5.84%

Task	Benckmark	Code Rep.+ Abs.	CodeBERT			GraphCodeBERT			PLBART			CodeT5			UniXcoder		
			PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last
❹	D4J V1.2	CR3_raw	**40/54**	34/48	31/46	**41/61**	39/53	39/53	**32/52**	25/39	25/41	**69/95**	53/77	53/77	**66/102**	52/76	54/82
	D4J V1.2	CR3_raw	**11.19%**	10.47%	10.83%	**10.11%**	8.84%	8.84%	**10.29%**	7.94%	7.76%	**17.69%**	12.64%	12.64%	**18.41%**	13.36%	14.08%

Task	Benckmark	Code Rep.+ Abs.	CodeBERT			GraphCodeBERT			PLBART			CodeT5			UniXcoder		
			PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last
❺	VulRD	CR3_raw	43.96%	**51.58%**	**51.58%**	43.61%	**53.28%**	53.22%	47.30%	59.85%	**60.08%**	52.93%	**62.78%**	**62.78%**	50.64%	62.08%	**62.19%**
		CR4_raw	31.07%	49.00%	**49.24%**	25.91%	41.85%	**41.97%**	46.19%	53.58%	**53.93%**	35.87%	55.86%	**55.92%**	40.62%	55.28%	**55.63%**

Task	Benckmark	Code Rep.+ Abs.	CodeBERT			GraphCodeBERT			PLBART			CodeT5			UniXcoder		
			PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last	PPL	BLEU	Last
❻	TFixD	CR3_raw	48.28%	52.39%	**52.83%**	48.53%	**52.33%**	**52.33%**	44.78%	**48.23%**	**48.23%**	50.44%	54.93%	**55.31%**	48.25%	**54.33%**	54.31%

Findings

RQ1: How do different design choices affect LLMCs' repair capability?

1. **Without Code Abstraction is more suitable.**
2. **Delicate Code Presentation matters!**
3. **Different repair scenarios have different checkpoint selection.**

Findings

RQ2: How well does the LLMC perform compared to the state-of-the-art approaches?

—> LLMC outperforms previous APR works in software bugs, security vulnerabilities, and programming errors.

Benchmark	Our Work					Baseline (Jiang et al. [21])										
	CodeBERT base	GraphCode base	PLBART base	CodeT5 base	UniXcoder base	PLBART base	CodeT5 base	CodeGen 350M	CodeGen 2B	CodeGen 6B	InCoder 1B	InCoder 6B	DL-based APR Tool CURE	DL-based APR Tool Reward	DL-based APR Tool Recoder	DL-based APR Tool KNOD
Defects4J V1.2	33/47	34/47	16/31	49/62	57/71	25/-	30/-	23/-	32/-	38/-	27/-	41/-	6/-	20/-	24/-	20/-
Defects4J V2.0	25/44	28/48	24/41	33/45	46/67	13/-	17/-	20/-	23/-	23/-	24/-	28/-	6/-	8/-	11/-	13/-

Benchmark	Our Work					Baseline (Fu et al. [17])	
	CodeBERT	GraphCode	PLBART	CodeT5	UniXcoder	VulRepair	VRepair
VulRepair	52.17%	54.16%	60.90%	64.71%	63.77%	44.67%	23.00%

Benchmark	Our Work					Baseline (Berabi et al. [24])		
	CodeBERT	GraphCode	PLBART	CodeT5	UniXcoder	TFix	CoCoNuT	SequenceR
TFix dataset	57.42%	56.45%	53.28%	60.62%	60.10%	49.30%	11.70%	17.90%

Findings

RQ3: What are the factors that limit the effectiveness of fine-tuning LLMCs?

1. **Loss of Pre-trained Knowledge.**
2. **Lack of Repair Ingredients and Long Sequence Problem.**
3. **Demanding Computing Resource and Large Model Size.**
4. **Long-tail and Small-sample Problem.**
5. **Difficulty from Multi-Hunk Fixes.**

Relation to Our Project

- Comprehensive, but more focus on Fine-tuning LLMC.
- The datasets mainly focus on Java Benchmark.
- Not consider the results by multiple interactions with users providing extra information.

InferFix: End-to-End Program Repair with LLMs over Retrieval-Augmented Prompts

Matthew Jin*, Syed Shahriar+, Michele Tufano*, Xin Shi*, Shuai Lu§, Neel Sundaresan*, Alexey Svyatkovskiy*

*Microsoft Redmond, WA, USA
+UCLA Los Angeles, CA, USA
§Microsoft Research Beijing, China

ESEC/FSE ’23, December 3–9, 2023, San Francisco, CA, USA

Motivation

- Addressing the high costs associated with identifying and repairing software defects in development.

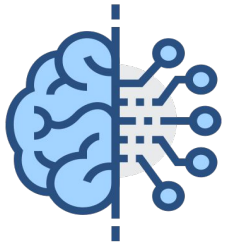
- Exploring the use of LLMs and static analyzers to automate and improve the efficiency of software defect repair.

Research Questions

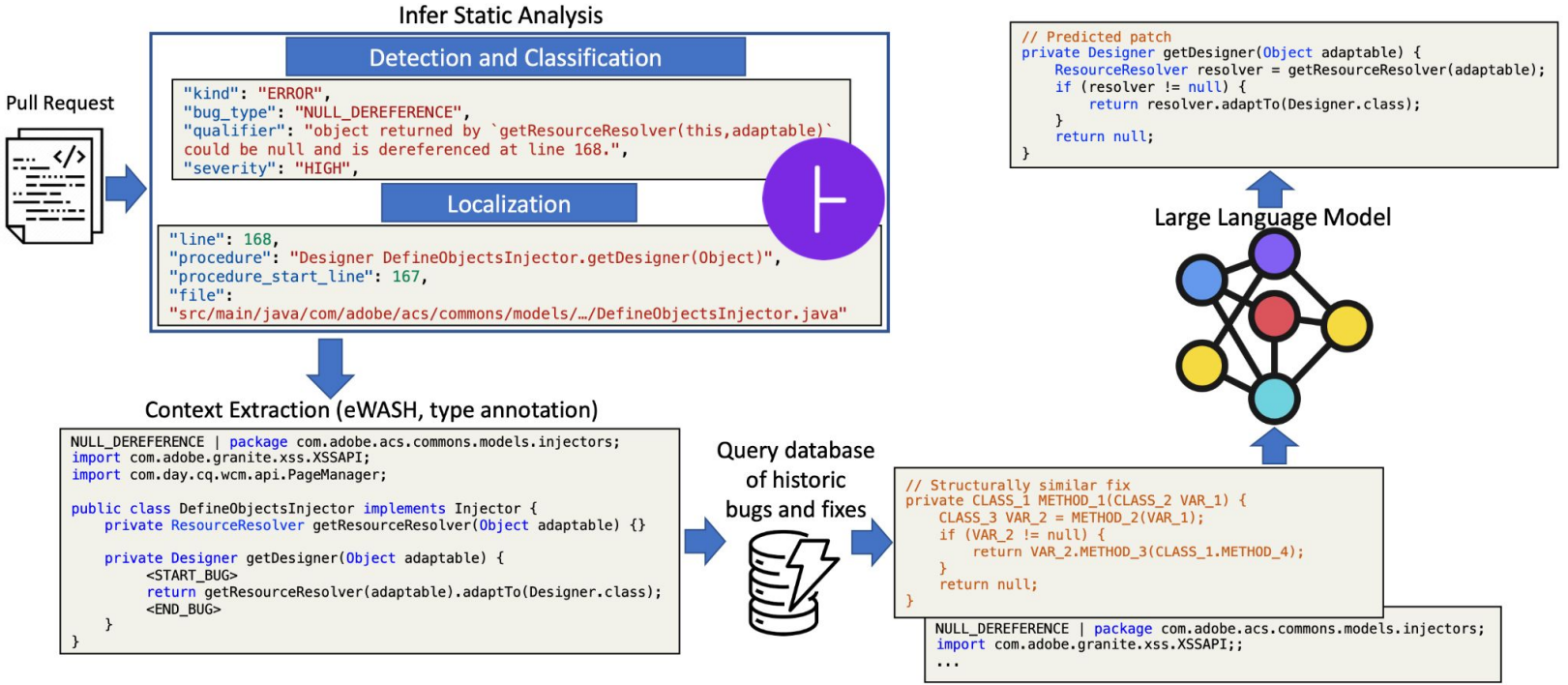
- **RQ1:** How effective are LLMs in Automated Program Repair?
- **RQ2:** Can the integration of static analyzers and LLMs enhance program repair?
- **RQ3:** What is the impact of Retrieval-Augmented prompts in program repair?

Datasets, Tools and Technologies

- **APR Tools -** CoCoNuT, DLFix, CURE
- **Dataset -** InferredBugs
- **Technologies -** Infer Static Analyzer, InferSharp



Methodology



Results

Table 2: Evaluation results for InferFix with basic prompt compared against LLM baselines

Approach	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
Demonstration (Codex)	20.3	30.1	25.3	29.1	19.0	16.7
Completion (Codex)	6.7	6.1	7.8	5.7	3.9	0.0
Instruction (Davinci)	40.5	22.2	53.8	19.7	41.3	33.3
InferFix (basic prompt)	49.7	58.1	60.0	51.9	64.4	70.0

Table 3: Evaluation results for InferFix demonstrating the impact of introducing the bug-type annotation in the prompt.

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (basic prompt)	49.7	58.1	60.0	51.9	64.4	70.0
InferFix (+ bug type)	52.3	60.4	63.1	53.3	67.9	72.5

Table 4: Evaluation results for InferFix showing the effect of adding bug location markers in the prompt.

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (bug type)	52.3	60.4	63.1	53.3	67.9	72.5
InferFix (+ localization)	53.5	61.4	64.4	53.9	69.6	75.0

Table 5: Evaluation results for InferFix showing the effect of adding eWASH extended context in the prompt.

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (localization)	53.5	61.4	64.4	53.9	69.6	75.0
InferFix (+ eWASH)	57.6	65.1	69.1	56.1	75.0	80.0

Table 6: Evaluation results for InferFix showing the effect of adding bug-fix hints.

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (eWASH)	57.6	65.1	69.1	56.1	75.0	80.0
InferFix (+ retrieved hints)	59.5	66.7	71.2	57.0	77.4	82.5

Table 7: Evaluation results for InferFix on the InferredBugs dataset compared against LLM baselines

Approach	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
Demonstration (Codex)	20.3	30.1	25.3	29.1	19.0	16.7
Completion (Codex)	6.7	6.1	7.8	5.7	3.9	0.0
Instruction (Davinci)	40.5	22.2	53.8	19.7	41.3	33.3
Finetuning (Codex)	49.7	58.1	60.0	51.9	64.4	70.0
InferFix	**59.5**	**66.7**	**71.2**	**57.0**	**77.4**	**82.5**

UCDAVIS

Relation to Our Project

- **Language Expansion:** Our project extends the scope of LLMs in program repair.
- **Contextual Analysis:** Echoes InferFix's use of augmented prompts by assessing the impact of additional context on LLMs in program repair, especially in the C/C++ domain.
- **Comparative Study:** Builds on InferFix's integration of LLMs and static analyzers by comparing this approach with traditional repair tools in C/C++, enriching the understanding of LLMs in software engineering.

Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT

Chunqiu Steven Xia and Lingming Zhang

University of Illinois Urbana-Champaign

arXiv:2304.00385

Submitted on Apr 1st, 2023 and has 104 citations on Google Scholar

Motivation

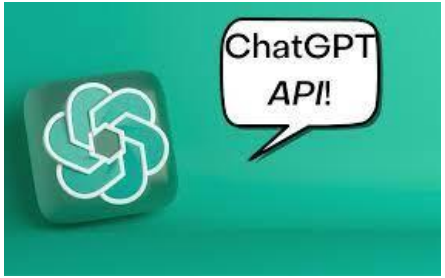
- Advancing Automated Program Repair (APR) through a conversation-driven approach
- Evaluating and benchmarking the performance of the proposed solution (ChatRepair)
- Assessing the effectiveness and generalizability of the proposed (conversational) paradigm

RQs

- **Research Questions**
  - **How does the performance of ChatRepair compare against the state-of-the-art techniques for APR?**
  - **How does ChatRepair perform when used in different repair scenarios?**
  - **What are the contributions of different components of ChatRepair in improving repair effectiveness?**

Tools, Technologies and Datasets

- **ChatRepair -** Implemented in Python using ChatGPT (gpt-3.5-turbo-0301)'s API
- **APR Tools -** 9 NMT-based or LLM-based + 12 traditional
- **Datasets -** Defects4j and Quixbugs



rjust/**defects4j**

A Database of Real Faults and an Experimental Infrastructure to Enable Controlled Experiments in Software Engineering Research

27 Contributors · 63 Issues · 629 Stars · 285 Forks

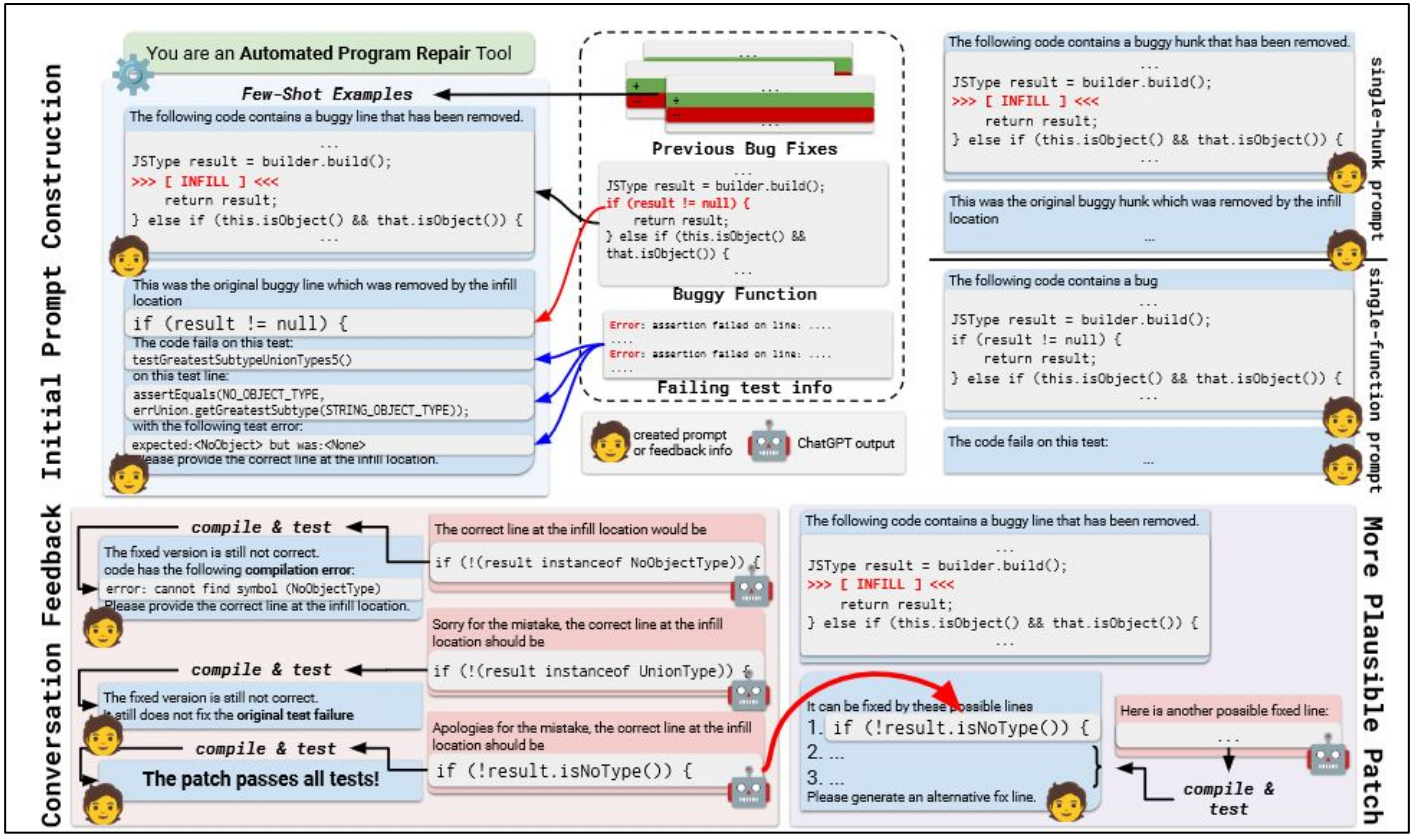
jkoppel/**QuixBugs**

A multi-lingual program repair benchmark set based on the Quixey Challenge

10 Contributors · 3 Issues · 75 Stars · 36 Forks

UCDAVIS

Methodology



Evaluation Metrics

- Plausible patches
- Correct Patches
- Tries
- Dollar cost

Results

Table 1: Correct fixes on Defects4j

Dataset	ChatRepair	BaseChatGPT	CodexRepair	AlphaRepair	SelfAPR	RewardRepair	Recoder	TBar	CURE	CoCoNuT
Chart	15	9	9	9	7	5	10	11	10	7
Closure	37	23	30	23	19	15	21	16	14	9
Lang	21	15	22	13	10	7	11	13	9	7
Math	32	25	29	21	22	19	18	22	19	16
Mockito	6	6	6	5	3	3	2	3	4	4
Time	3	2	3	3	3	1	3	3	1	1
D4J 1.2	**114**	80	99	74	64	50	65	68	57	44
D4J 2.0	**48**	25	31	36	31	25	11	8	-	-

Table 2: Correct fixes on QuixBugs

QuixBugs	Chart Repair	Base ChatGPT	Codex Repair	Alpha Repair	CoCoNuT
Python	40	40	40	27	19
Java	40	40	38	28	13

Table 3: Correct fixes using three repair settings

Tools	D4J 1.2			Quixbugs-Py			Quixbugs-J		
	SL	SH	SF	SL	SH	SF	SL	SH	SF
ChatRepair	57	79	76	39	40	40	36	37	39
BaseChatGPT	41	55	45	38	37	35	33	36	39
CodexRepair	39	62	63	39	39	37	34	34	32

ChatRepair outperforms baseline APR tools on nearly all datasets in all repair settings/scenarios.

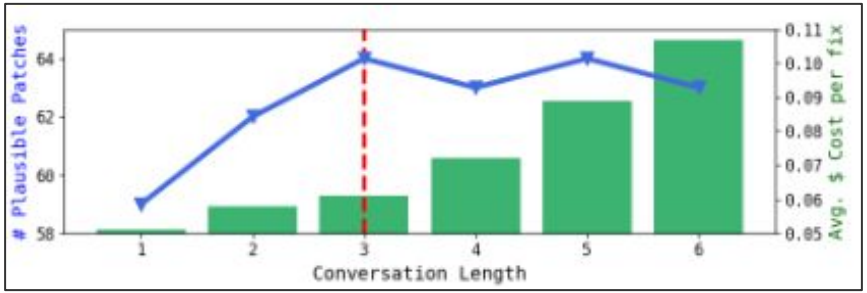
Results

Table 4: Initial prompt variations

Initial Prompt	#P	Avg. # tries	Avg. $
BasePrompt	55	22.53	$0.069
TestName+ErrMsg	59	22.47	$0.072
TestName+ErrMsg+FailLine	**64**	**21.86**	**$0.061**
TestName+ErrMsg+TestBody	61	23.42	$0.083
`You are a helpful assistant`	**64**	24.17	$0.074
`You are an APR tool`	**64**	**21.86**	**$0.061**
0-shot	64	21.86	**$0.061**
1-shot	**65**	9.91	$0.072
2-shot	**65**	**9.87**	$0.085

Table 5: Feedback response variations

Feedback Response	#P	Avg. # tries	Avg. $
BaseFeedback	58	23.12	$0.071
TestName+ErrMsg	61	22.48	$0.073
TestName+ErrMsg+FailLine	62	24.71	$0.074
Dynamic	**64**	**21.86**	**$0.061**



Effects of ChatRepair components (initial prompts, feedback responses, and conversation lengths) on efficiency.

Relation to Our Project

- LLM-Based APR Comparison
- Automated Conversation-Driven Approach
- Real-world Bug Dataset
- Evaluation Metrics Alignment

Conclusion

Systematically explore the **capabilities** of 3 generative AI chatbots based on LLM to *fix various bugs* in real-world C/C++ projects

RQ1: How effective are LLMs in repairing different types of bugs in C/C++ and does their effectiveness vary under different repair scenarios?

RQ2: Do LLMs perform better with additional context information in extra rounds of interactions with the user?

- Fully automated prompting method to (iterative) query LLM models.
- Multiple Repair Scenarios: Bug Types + single/multi hunk.
- Metric: Plausible/Correct Patches, Repair Accuracy, Response Time, Tries.

RQ3: How do LLMs compare with existing traditional C/C++ APR tools?

- Utilize at least one APR tool for C/C++ as the baseline model.

Thank you!