Proceedings of the 2nd International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/113/2024.18303

Large Language Models Meet Automated Program Repair: Innovations, Challenges and Solutions

Yiting Tang

Northwest Minzu University, Lanzhou, China

{tangyt}p221513249@stu.xbmu.edu.cn

Abstract. As the field of Automated Program Repair (APR) continues to evolve, traditional Neural Program Repair (NPR) methods, while successful in low-resource computing scenarios, still confront numerous challenges, including the demand for extensive training data, the limited generality of specially designed networks, and a lack of robustness. In recent years, Large Language Models (LLMs) have demonstrated remarkable efficacy in downstream code-related tasks, thanks to their potent comprehension and text generation capabilities, gradually emerging as pivotal tools in automated program repair. Compared to NPR techniques, LLM-based APRs exhibit superior repair performance and enhanced generality, leading to their increasing adoption in APR tasks. Currently, the performance of zero-shot LLM-based APRs has surpassed that of NPR. LLM-based APRs have issues, such as excessive fine-tuning costs, data leakage concerns, and a shortage of domain-specific knowledge. This paper aims to review and summarize the latest advancements in LLM-based APRs from the perspectives of innovation, challenges, and solutions, providing researchers with profound insights and future directions.

Keywords: Automated Program Repair, Neural Program Repair, Large Language Model, Software Quality Assurance, Survey.

1. Introduction

Automated Program Repair (APR) has gradually emerged as a significant research topic in software engineering and artificial intelligence. In recent years, with the development of deep learning technologies, Neural Program Repair (NPR), as an emerging APR method, has increasingly attracted the attention of researchers. Fig 1 is a workflow of the learning-based automated program repair tool. NPR typically leverages deep neural networks (DNNs) within a neural machine translation framework to transform buggy code into correct code, essentially treating the program repair task as a machine translation task where buggy code is "translated" into correct code. During training, NPR systems aim to maximize the probability distribution function of actual patches, continuously optimizing the model to enhance repair performance. The core advantage of NPR lies in its ability to learn complex repair patterns from numerous bug-fix pairs, thereby outperforming traditional APR methods in terms of performance. Recorder is the first NPR system to surpass traditional APR techniques on the Defects4J dataset, exhibiting superior repair performance compared to the then-current State-of-the-Art (SOTA) technique, TBar. Despite the significant progress made by NPR in repair performance, numerous challenges still need to be overcome, including the demand for extensive training data, limited model generalization capabilities, and reliance on specialized network designs[1].

DOI: 10.54254/2755-2721/113/2024.18303

Recently, the advent of Large Language Models (LLMs) has demonstrated remarkable capabilities in various tasks within software engineering. Compared to traditional NPR, LLM-based APR excels in repair performance and flexibility, particularly in zero-shot learning scenarios, surpassing many custom-trained NPR models[2-4]. However, LLM-based APR also encounters new challenges, such as high fine-tuning costs, risks of data leakage, a lack of domain-specific knowledge, and insufficient adaptability in complex repair scenarios [5-7]. These challenges indicate that continuous optimization and improvement are necessary for systems to harness the potential of LLM-based APR in practical applications.

- The contributions of this paper are as follows:To the best of our knowledge, our paper is the first to provide a comprehensive review of the LLM-based Automated Program Repair (APR) domain from the perspectives of innovation, challenges, and solutions, offering a novel viewpoint to this field.
- We have investigated eight state-of-the-art LLM-based APR systems, compared their performance, analyzed their innovations and challenges, and proposed solutions to address those challenges.
- We outline future directions for LLM-based APR systems.

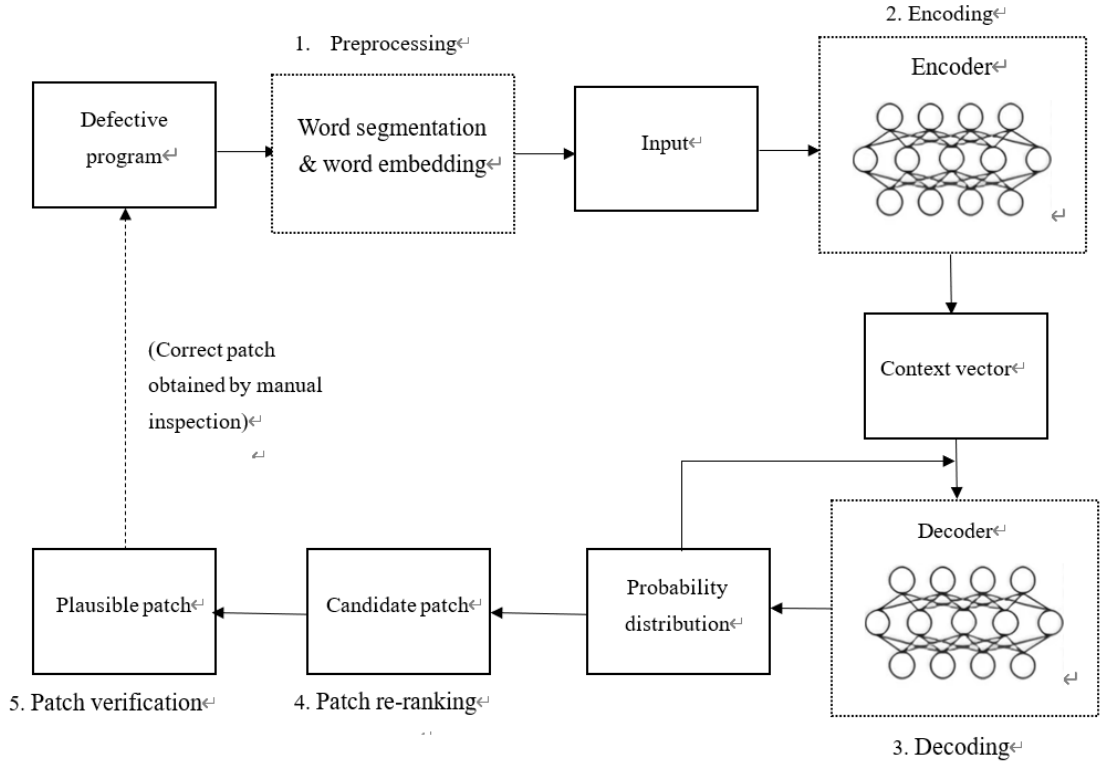

Figure 1. Workflow of the learning-based automated program repair tool.

2. Benchmarks and Metrics

2.1. Benchmarks

In Automated Program Repair (APR) tasks, benchmarks evaluate and compare the effectiveness of different repair techniques.

2.1.1. Java Language Benchmark

Defects4J (v1.2/v2.0): This is a widely used benchmark dataset for Java program repair. It contains Java program bugs extracted from multiple open-source projects—the—total number of bugs covered by Defects4J v1.2 and v2.0 amounts to 835.

DOI: 10.54254/2755-2721/113/2024.18303

HumanEval-Java: This is a relatively new Java benchmark dataset for evaluating program repair techniques. The dataset consists of 164 bugs, created by transforming the language of HumanEval and subsequently injecting bugs.

2.1.2. Python Language Benchmark

QuixBugs: QuixBugs is a benchmark dataset for Python, encompassing 40 bugs. These bugs originate from various Python open-source projects like Django and Flask. This benchmark dataset is specifically designed to assist researchers in testing and comparing the effectiveness of Python program repair techniques.

BugsInPy: This is another benchmark dataset for Python, containing bugs extracted from multiple Python projects. While the volume of data and specific sources of bugs in BugsInPy may vary, it provides a rich array of bug examples for researching Python program repair.

2.1.3. C Language Benchmark

CodeFlaw: CodeFlaw is a benchmark dataset designed for C language programs, with its bugs originating from various C language open-source projects such as SQLite and libpng. Although CodeFlaw is suitable for testing C language program repair techniques, due to its relatively small size, it may only comprehensively cover some common types of bugs found in C language programs.

2.2. Metrics

The primary evaluation metrics for Automated Program Repair (APR) tasks include Incorrect Patch, Plausible Patch, and Correct Patch [8-10]. An Incorrect Patch is a generated patch that fails to repair the bug in the code correctly or introduces new errors. A Plausible Patch addresses the code defect but has not been thoroughly tested and verified. A Correct Patch, on the other hand, effectively repairs the defect without introducing any new issues [11].

3. LLM-Based APR review

Currently, there is a diverse range of Automated Program Repair (APR) methods based on Large Language Models (LLM), including FitRepair, TypeFix, InferFix, RepairAgent, FixAgent, ThinkRepair, Srepair, and ChatRepair [12-19]. These methods have unique characteristics regarding model architecture, technical implementation, and application effects, providing new ideas and approaches for automated program repair. Fig 2 is an overview of the standard framework of LLM-Agent-based APRs.

Figure 2. The standard framework of LLM-Agent-based APRs.

3.1. FitRepair

FitRepair leverages program synthesis techniques, integrating test cases and program specifications to generate repair patches. It ensures that the generated code passes all relevant tests. Its objective is to enhance software reliability by providing high-quality automated repairs.

DOI: 10.54254/2755-2721/113/2024.18303

3.2. TypeFix
TypeFix addresses type mismatch issues by generating corresponding type repair patches. It analyzes type information within the program to ensure that the repairs accurately resolve type-related defects.

3.3. InferFix
InferFix generates repair patches by reasoning about the program's semantics and structure, aiming to improve repair accuracy. This method leverages contextual information to understand code intent, enhancing the repairs' effectiveness.

3.4. RepairAgent
RepairAgent employs deep learning models to analyze code defects and generate repair suggestions. By learning from historical repair cases, this tool applies that knowledge to new bug detection tasks, enhancing the intelligence level of automated repairs.

3.5. FixAgent
FixAgent integrates multiple repair strategies to provide flexible repair solutions. It comprehensively considers various contextual information when generating patches, enhancing the repairs' effectiveness and adaptability.

3.6. ThinkRepair
ThinkRepair leverages program analysis and reasoning techniques to generate high-quality repair code, emphasizing code readability and maintainability. This tool aims to provide sustainable repair solutions and reduce subsequent maintenance costs.

3.7. SRepair
SRepair automatically generates repair patches through symbolic execution and program analysis techniques, mainly targeting complex program errors. Its design objective is to enhance the reliability and precision of repairs in addressing a wide range of defects.

3.8. ChatRepair
ChatRepair, based on a dialogue generation model, provides interactive code repair suggestions. Through dialogue, users can adjust and optimize the repair solutions to meet specific needs and contexts.

Table 1. State-of-the-art LLM-based APR.

		InferFix	FitRepair	RepairAgent	FixAgent	ThinkRepair	Repair	PyTy	TypeFix	ChatRepair
Approach Design	Model Name	Codex	CodeT5	GPT-3.5	GPT-4	GPT-3.5	GPT-3.5 MigiCoder	TFix(T5)	CodeT5	ChatGPT
	Model Architecture	Decoder-only	Encoder-Decoder	Decoder-only	Decoder-only	Decoder-only	Decoder-only	Encoder-Decoder	Encoder-Decoder	Decoder-only
	Technique	Retrieval	Fine-tuning	Retrieval	Ensemble	CoT、Ensemble、Multi-round Dialogues、Fine-tuning	Multi-round Dialogues、CoT	Fine-tuning	Fine–tuning、Retrieval	Multi-round Dialogues、Agent
Performance	Defects4J(v1.2/v2.0)	-	-	164(74/90)	-	98/-	332(300/32)	-	-	162(114/48)
	HumanEval-Java	-	-	-	-	-	-	-	-	-
	QuixBugs	-	-	-	79	39	-	-	-	39
	BugsInPy	-	-	-	-	-	-	-	26	-
	CodeFlaw	-	-	-	-	-	-	-	-	-

DOI: 10.54254/2755-2721/113/2024.18303

4. Innovations

4.1. Reduced Training Requirements for LLM-based APR:

Compared to traditional Neural Program Repair (NPR), Automated Program Repair (APR) based on Large Language Models (LLM) offers significant advantages in terms of training requirements. LLM-based APR typically does not require additional training. For instance, AlphaRepair treats repair tasks as fill-in-the-blank problems, similar to the masked language model task in CodeBERT, thus achieving excellent results [20]. Srpeiar employs context completion techniques, while FixAgent leverages the GPT-3.5 API without needing extra processes. ThinkRepair, on the other hand, enhances precision through logical reasoning[21-22]. InferFix and FitRepair quickly generate repair solutions through semantic analysis and model versatility. RepairAgent and PyTy fully exploit the potential of pre-trained models in intelligent interaction and language understanding—TypeFix and ChatRepair advance code repair by utilizing open-source models and dialogue formats.

4.2. Simplified Network Design

In contrast to traditional Neural Program Repair (NPR) methods, LLM-based APR eliminates the need for complex network design. FixAgent and Srpeiar directly utilize GPT-3.5, while AlphaRepair and TypeFix rely on open-source models such as CodeBERT and CodeT5, all based on the Transformers architecture. Comparatively, NPR methods like Recorder, DlFix, and CoCoNuT require intricate network structures, such as tree-structured deep neural networks or the integration of multiple CNNs, for hierarchical representation and processing.

4.3. Adoption of Advanced Techniques

LLM-based APR actively incorporates a more significant number of advanced techniques. Compared to NPR, which typically relies on training or fine-tuning, LLM-based APR utilizes additional generated information to aid reasoning, incorporating techniques such as Chain-of-Thought (CoT), agents, retrieval, and multi-round dialogues [23-25].

4.3.1. InferFix

InferFix leverages semantic and structural analysis, combined with contextual information and Retrieval techniques, to retrieve relevant information from vast code repositories, thereby enhancing the accuracy of repairs [24].

4.3.2. FitRepair

FitRepair integrates model ensemble techniques, explicitly incorporating a CodeT5 Base, two fine-tuned CodeT5 models, and a retrieval-enhanced CodeT5 to improve repair capabilities[25].

4.3.3. RepairAgent

RepairAgent generates repair solutions by analyzing error types, context, and programming languages and retrieving similar historical cases[26].

4.3.4. FixAgent

FixAgent combines multiple repair strategies with Ensemble techniques, integrating different models and methods to enhance performance and reliability [27].

4.3.5. ThinkRepair

ThinkRepair employs CoT (Chain-of-Thought) techniques to analyze erroneous code deeply, comprehend its logic and semantics, uncover the nature of errors, and propose repair suggestions[28].

4.3.6. SRepair

SRepair augments the repair process by generating additional information. It initially uses GPT-3.5 to produce detailed error reasons and repair suggestions for the buggy function. Subsequently, the buggy

DOI: 10.54254/2755-2721/113/2024.18303

function, along with these detailed reasons and tips, is input into MigiCoder-7B to generate higher-quality patches[29].

4.3.7. TypeFix

TypeFix specializes in resolving type mismatch issues by analyzing type information to generate precise repair patches. It employs Fine-tuning techniques based on pre-trained models to fine-tune for specific languages and type errors, enhancing its understanding of type information [30]. Using Retrieval techniques, it searches code repositories to retrieve relevant code snippets and repair experiences, analyzing error characteristics to find similar error cases and corresponding repair methods.

4.3.8. ChatRepair

ChatRepair employs a dialogue generation model to enable interactive code repair and supports multi-round dialogues to refine repair solutions [31]. Users can provide contextual information and repair requirements, which the system analyzes and adjusts to generate repair code that meets the specified needs. ChatRepair can understand user requirements and reasoning through error information and propose potential repair methods.

5. Challenges and solutions

5.1. Challenge 1: Data Leakage

Data leakage is a significant challenge that must be considered when using large language models. Closed-source models, such as GPT-3.5 and GPT-4, have inaccessible training data, making it difficult to detect data leakage and design mitigation strategies. Although the training data for post-open-source models is accessible, the vast amount of data makes detecting and mitigating data leakage equally challenging [5-6].

5.2. Solution Strategies

5.2.1. Data Filtering + Re-training

"An Empirical Study on Data Leakage Issues in the Field of Neural Program Repair" proposes new data collection, filtering, and partitioning strategies to construct a clean dataset and re-train the NPR (Neural et al.) model to mitigate data leakage.

5.2.2. Detecting Memorization in Base Models

"Unveiling Memorization in Code Models" (ICSE 2023) designs different detection methods based on whether the training data is open-source. Models with open-source training data employ Type-1 clone detection to determine if the outputs are identical to code snippets in the training data. For models with closed-source training data, they measure the model's level of memorization for the outputs using the perplexity (PPL) of the output sequences.

5.3. Challenge 2: High Overhead

Using Large Language Models (LLMs) for Automated Program Repair (APR) presents a significant challenge in terms of high overhead, including the GPU requirements for open-source LLMs and the costs associated with API calls for closed-source models [32].

5.4. Solution Strategies

5.4.1. Reducing the Number of Candidate Patches

According to "StandUp4NPR: Standardizing SetUp for Empirically Comparing Neural Program Repair Systems" (Zhong et al., 2022), when the number of candidate patches reaches approximately 170, the model can achieve 90% of its optimal performance. Therefore, it is appropriate to reduce the number of candidate patches to lower overhead [33]. In practical applications, appropriately reducing the number

DOI: 10.54254/2755-2721/113/2024.18303

of candidate patches can decrease the computational resources and time required for generating and validating patches without significantly affecting the repair effectiveness.

5.4.2. Adoption of Advanced Techniques such as Self-Correction

As demonstrated in "Large Language Models Have Intrinsic Self-Correction Ability" (Liu et al., 2024), Self-Correction technology can effectively enhance the reasoning capabilities of LLMs. When applied in LLM-based APR, it is crucial to use unbiased prompts and turn off relevant hyperparameters [10]. Specifically, unbiased prompts should be used, and hyperparameters related to model samplings, such as do_sample, temperature, top_k, and top_p, should be turned off. This approach prevents the model from being influenced by unreasonable sampling when generating patches, thereby improving the accuracy and reliability of the patches while reducing unnecessary computational overhead. By appropriately applying Self-Correction technology, it is possible to enhance model performance while lowering overhead.

6. Conclusion and future

This paper presents an overview of the development history, innovative achievements, and current challenges faced by Automated Program Repair (APR) technologies based on large language models (LLMs) in recent years. LLMs have demonstrated advantages over traditional Neural Program Repair (NPR) approaches in various repair scenarios, highlighting their potential to cater to diverse program repair needs. However, it must be noted that issues such as data leakage and high computational costs still need to improve the advancement of these technologies.

Future research efforts should address these challenges. Firstly, there is a need to develop more effective methods for detecting and mitigating data leakage, which includes refining data filtering techniques and enhancing data leakage assessment methodologies. In program repair applications, technological breakthroughs are required to enable repairs at the project and repository levels, tackling complex systems and cross-module issues. Furthermore, addressing program repairs for low-resource languages, such as C# and Rust, and those involving intricate logic represents a crucial direction for future research. This necessitates advancements in LLMs' capability to comprehend and process complex algorithms and control flows.

Research should explore more efficient algorithms and optimization techniques to reduce computational overhead[34-35].For instance, reducing the number of candidate patches and refining self-correction techniques can enhance cost-effectiveness and efficiency. Further architectural innovations and training strategies may enhance LLMs' ability to handle extended contexts. Through these concerted efforts, LLM-based APR systems hold the promise of successful application in broader and more complex scenarios, pushing the boundaries of automated program repair technology.

References

[1] Wenkang Zhong, Hongliang Ge, Hongfei Ai, Chuanyi Li, Kui Liu, Jidong Ge, and Bin Luo. (2022). StandUp4NPR: Standardizing SetUp for Empirically Comparing Neural Program Repair Systems. In 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), Rochester, MI, USA, October 10-14, 2022, 13 pages. https://doi.org/10.1145/3551349.3556943

[2] Nan Jiang, Thibaud Lutellier, and Lin Tan. (2021). Cure: Code-Aware Neural Machine Translation for Automated Program Repair. In 43rd IEEE/ACM International Conference on Software Engineering (ICSE, 2021), Madrid, Spain, May 22-30, 2021, 1161-1173. https://doi.org/10.1109/ICSE43902.2021.00107

[3] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. (2019). An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. ACM Transactions on Software Engineering and Methodology.

DOI: 10.54254/2755-2721/113/2024.18303

[4] He Ye, Matias Martinez, Thomas Durieux, and Martin Monperrus. (2021). A Comprehensive Study of Automated Program Repair on the QuixBugs Benchmark. Journal of Systems and Software.

[5] Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, Donggyun Han, and David Lo. (2024). Unveiling Memorization in Code Models. In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), Lisbon, Portugal, April 14-20, 2024, 13 pages. https://doi.org/10.1145/3597503.3639074

[6] Yihong Dong, Xue Jiang, Huanyu Liu, Zhi Jin, Bin Gu†, Mengfei Yang‡, and Ge Li*. (2024). Generalization or Memorization: Data Contamination and Trustworthy Evaluation for Large Language Models. arXiv preprint arXiv:2402.15938v3.

[7] Wenkang Zhong, Chuanyi Li, Jidong Ge, and Bin Luo. (2022). Neural Program Repair: Systems, Challenges, and Solutions. arXiv preprint arXiv:2202.10868.

[8] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. (2023). InferFix: End-to-end program repair with LLMs over Retrieval - retrieval-augmented prompts. In Proceedings of ACM Conference (Conference'17). ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[9] Xin Yin, Chao Ni, Shaohua Wang, Zhenhao Li, Limin Zeng, and Xiaohu Yang. (2024). ThinkRepair: Self-Directed Automated Program Repair. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), Vienna, Austria, September 16–20, 2024, 13 pages. https://doi.org/10.1145/3650212.3680359

[10] Dancheng Liu∗, Amir Nassereldine∗, Ziming Yang∗, Chenhui Xu, Yuting Hu, Jiajie Li, Utkarsh Kumar, Changjae Lee, Jinjun Xiong†. (2024). Large Language Models have Intrinsic Self-Correction Ability—arXiv preprint arXiv:2406.15673v1.

[11] He Ye, Matias Martinez, and Martin Monperrus. (2021). Automated patch assessment for program repair at scale. Empir. Softw. Eng. 26(2), 20. https://doi.org/10.1007/s10664 - 020 - 09920 - w

[12] Chunqiu Steven Xia and Lingming Zhang. (2023). Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT—arXiv preprint arXiv:2304.00385.

[13] Chunqiu Steven Xia and Lingming Zhang. (2023). Revisiting the Plastic Surgery Hypothesis via Large Language Models. arXiv preprint arXiv:2303.10494.

[14] Yun Peng, Shuzheng Gao, Cuiyun Gao, Yintong Huo, and Michael R. Lyu. (2023). Domain Knowledge Matters: Improving Prompts with Fix Templates for Repairing Python Type Errors. In Proceedings of 46th International Conference on Software Engineering (ICSE 2024).

[15] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. (2023). InferFix: End-to-end program repair with LLMs over Retrieval - retrieval-augmented prompts. In Proceedings of ACM Conference (Conference'17).

[16] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. (2024). RepairAgent: An Autonomous, LLM - Based Agent for Program Repair. arXiv preprint arXiv:2403.17134v1.

[17] Cheryl Lee, Chunqiu Steven Xia, Jen - tse Huang, Zhouruixin Zhu, Lingming Zhang, and Michael R. Lyu. (2024). A Unified Debugging Approach via LLM - Based Multi-Agent Synergy.

[18] Xin Yin, Chao Ni, Shaohua Wang, Zhenhao Li, Limin Zeng, and Xiaohu Yang. (2024). ThinkRepair: Self-Directed Automated Program Repair. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), Vienna, Austria, September 16–20, 2024.

[19] Mingyuan Wu, Jiahong Xiang, Xiaoyang Xu, Fanchu Kong, Haotian Zhang, and Yuqun Zhang. (2024). How Far Can We Go with Practical Function-Level Program Repair? arXiv preprint arXiv:2404.12833v1.

[20] Chunqiu Steven Xia and Lingming Zhang. (2022). Less Training, More Repairing Please: Revisiting Automated Program Repair via Zero-Shot Learning. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, November 14 - 18, 2022.

DOI: 10.54254/2755-2721/113/2024.18303

[21] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. (2023). InferFix: End-to-end program repair with LLMs over Retrieval - retrieval-augmented prompts. In Proceedings of ACM Conference (Conference'17).

[22] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. (2024). RepairAgent: An Autonomous, LLM - Based Agent for Program Repair. arXiv preprint arXiv:2403.17134v1.

[23] Chunqiu Steven Xia and Lingming Zhang. (2023). Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT—arXiv preprint arXiv:2304.00385.

[24] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. (2023). InferFix: End-to-end program repair with LLMs over Retrieval - retrieval-augmented prompts. In Proceedings of ACM Conference (Conference'17).

[25] Chunqiu Steven Xia and Lingming Zhang. (2023). Revisiting the Plastic Surgery Hypothesis via Large Language Models. arXiv preprint arXiv:2303.10494.

[26] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. (2024). RepairAgent: An Autonomous, LLM - Based Agent for Program Repair. arXiv preprint arXiv:2403.17134v1.

[27] Cheryl Lee∗, Chunqiu Steven Xia†, Jen - tse Huang∗, Zhouruixin Zhu‡, Lingming Zhang†, and Michael R. Lyu∗. (2024). A Unified Debugging Approach via LLM - Based Multi-Agent Synergy.

[28] Xin Yin, Chao Ni, Shaohua Wang, Zhenhao Li, Limin Zeng, and Xiaohu Yang. (2024). ThinkRepair: Self-Directed Automated Program Repair. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24), Vienna, Austria, September 16–20, 2024.

[29] Mingyuan Wu, Jiahong Xiang, Xiaoyang Xu, Fanchu Kong, Haotian Zhang, and Yuqun Zhang. (2024). How Far Can We Go with Practical Function-Level Program Repair? arXiv preprint arXiv:2404.12833v1.

[30] Yun Peng, Shuzheng Gao, Cuiyun Gao, Yintong Huo, and Michael R. Lyu. (2023). Domain Knowledge Matters: Improving Prompts with Fix Templates for Repairing Python Type Errors. In Proceedings of 46th International Conference on Software Engineering (ICSE 2024).

[31] Chunqiu Steven Xia and Lingming Zhang. (2023). Keep the Conversation Going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT—arXiv preprint arXiv:2304.00385.

[32] Mingyuan Wu, Jiahong Xiang, Xiaoyang Xu, Fanchu Kong, Haotian Zhang, and Yuqun Zhang. (2024). How Far Can We Go with Practical Function-Level Program Repair? arXiv preprint arXiv:2404.12833v1.

[33] Wenkang Zhong, Hongliang Ge, Hongfei Ai, Chuanyi Li, Kui Liu, Jidong Ge, and Bin Luo. (2022). StandUp4NPR: Standardizing SetUp for Empirically Comparing Neural Program Repair Systems. In 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), Rochester, MI, USA, October 10–14, 2022.

[34] Li Qing - Yuan, ZHONG Wen - Kang, LI Chuan - Yi, GE Ji - Dong, LUO Bin. (2024). Empirical Study on the Data Leakage Problem in Neural Program Repair. Journal of Software, 35 (7), 3071–3092.

[35] Mingyuan Wu, Jiahong Xiang, Xiaoyang Xu, Fanchu Kong, Haotian Zhang, and Yuqun Zhang. (2024). How Far Can We Go with Practical Function-Level Program Repair? arXiv preprint arXiv:2404.12833v1.