The ET Program Repair Tool for Java

Yuan-An Xiao, Qihao Zhu, Yingfei Xiong*
{xiaoyuanan,zhuqh,xiongyf}@pku.edu.cn
Key Laboratory of High Confidence Software Technologies (Peking University), MoE; School of Computer Science, Peking University
Beijing, China

ABSTRACT

This is the ET tool participating in APR-Comp 2024. It is an end-to-end program repair approach that performs fault localization, patch generation, and patch validation.

ET is ranked as #1 in the *Functional Errors - Java* track.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Software evolution**.

KEYWORDS

Automated Program Repair

ACM Reference Format:
Yuan-An Xiao, Qihao Zhu, Yingfei Xiong. 2024. The ET Program Repair Tool for Java. In *2024 ACM/IEEE International Workshop on Automated Program Repair (APR '24), April 20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3643788.3648016

1 INTRODUCTION

ET is a Java program repair tool submitted to both the *Functional Errors - Java* track and the *AI Generated Code - Java* track in the APR-Comp 2024 competition [8].

The name ET comes from the initials of ExpressAPR [13, 14] and Tare [17], two major technologies used in this tool. The tool is publicly available on Zenodo with the DOI `10.5281/zenodo.8405059` [15].

In this paper, we briefly introduce the usage and the internal repair workflow of ET. We also report the results of ET in the competition.

2 USAGE

ET is distributed with a pre-built image on DockerHub (`xmcp/et:240114.1` [11]) and integrated into the Cerberus [6] program repair framework through Pull Request #115 [12]. It should work on any Linux computer under the specification of the two tracks in APR-Comp.

*Corresponding author.

APR '24 , April 20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0577-9/24/04
https://doi.org/10.1145/3643788.3648016

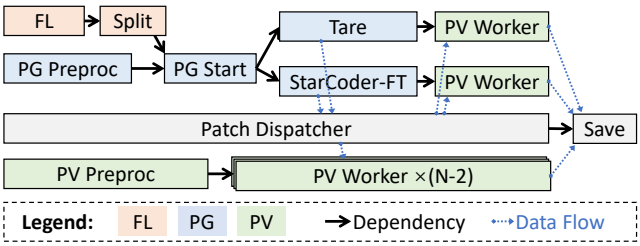

Figure 1: Workflow of ET

The user should prepare the benchmark as a Cerberus driver (e.g., `aprcompfuncjava`), and run the below Cerberus command line to evaluate ET against this benchmark:

```
$ python3 Cerberus.py -use-gpu -cpu-count 8 -task repair -tool et -benchmark aprcompfuncjava
```

ET will output at most five plausible patches in the `output/artifacts/(bug-id)/patches` directory, as required by the competition. Patch files are named as n`.diff`, where $1 \leq n \leq 5$. All files follow the unified diff format [7]. The directory will be empty if no plausible patches are found. ET will also store other information in the output directory, including candidate patches, fault localization results, and logs, for troubleshooting.

3 REPAIR WORKFLOW

3.1 Outline

As a generate-and-validate APR approach, ET internally takes three steps to repair a bug.

1. *Fault Localization (FL)*, where it runs the test suite to find suspicious lines;
2. *Patch Generation (PG)*, where two neural-network models generate candidate patches against each suspicious line;
3. *Patch Validation (PV)*, where it repeatedly runs the test suite against patched code and finds plausible patches.

The overall workflow of ET is shown in Figure 1. We write a main script in Python to glue all integrated procedures among three steps together. To maximize performance in the eight-core environment, multiple procedures can run in parallel whenever there are no dependencies. Each rectangle block refers to a procedure, and each arrow refers to a dependency between procedures. Procedures in different steps are drawn in different colors for ease of reading.

In the rest of this section, we go through technologies and libraries used in ET.

3.2 Fault Localization

For fault localization, we calculate the Ochiai [1] score with a tweaked version of the Flacoco [4] tool.

Our tweaked version of Flacoco fixes several bugs we have encountered on the demo benchmark of both tracks, where there may be nested test classes, ignored / disabled test cases, and test methods with whitespace characters[1]. We also tweaked it to output the list of failing test methods for patch validation.

After Flacoco outputs the list of suspicious locations, we truncate it to at most 150 lines for the functional track or 75 lines for the AI track. To avoid exhausting the VRAM in the patch generation step, the list is then split into several chunks, each chunk including no more than 30 lines.

3.3 Patch Generation

For patch generation, we run two independent models in parallel: Tare [17] and fine-tuned StarCoder [9]. Both models take a chunk from the fault localization step at a time, search for possible modifications for each suspicious line in the chunk, and report them back to the patch dispatcher thread in the main script via a `PUSH-PULL` ZeroMQ socket [5].

3.3.1 Tare. Tare is a type-aware neural model for program repair. Since the previous tools do not consider the constraints of code captured by a set of typing rules, the previous models often generate untypable patches. Tare learns the type information of the patches via learning the individual typing rules. To encode an individual typing rule, Tare proposes three novel techniques: (1) a novel type of grammar, T-Grammar, that integrates the type information into a standard grammar, (2) a novel representation of code, T-Graph, which integrates the key information needed for type-checking an AST, and (3) a novel transformer-based neural network, that encodes the T-Graph and generates the patches guided by T-Grammar. Integrated with these techniques, Tare tends to generate more typable patches and repairs the most bugs in Defects4J v1.2.

Tare requires a preprocessing step that reads all identifier names in the project. We run this preprocessing step in parallel with the fault localization step to minimize time usage.

3.3.2 StarCoder-FT. As the large pre-trained models show promising performance on several code-related tasks, we develop a program repair model based on the large pre-trained model, StarCoder Base-1b [9]. To enhance the repair ability of the base model, we use the collected patch datasets [10, 16] to finetune the base model for 3 epochs. We add two special tokens, *buggyline_begin* and *buggyline_end*, to mark the faulty position. During inference, we use the sampling generation approach with the temperature set to 0.8.

3.4 Patch Validation

For patch validation, we choose ExpressAPR [13, 14], an efficient patch validation approach. Considering that the benchmark in APR-Comp uses Maven [2] for project management, and some projects require the submodule feature that is currently not supported by the replication package of ExpressAPR, we re-implemented techniques from the ground up. We also use mvnd [3], a daemonized version of Maven, to further increase the patch validation speed.

[1]We are surprised to find that method names can contain whitespace characters in some JVM languages like Kotlin. This is observed in the lettuce-core project in the benchmark of the Functional Errors - Java track.

```
   public double constrain(double value) {
     if (contains(value)) {
       return value;
     }
     if (value > this.upper) {
       return this.upper;
     }
-    return this.lower;
+    return (value < (lower)) ? lower :
+      (value > (upper)) ? upper : value;
   }
```

Figure 2: A patch generated by ET for the JFreeChart project

The patch validation step requires a preprocessing procedure to set up mvnd and make a distinct copy of the project directory for each of the N patch validation workers (N is the number of CPU cores available: 8 for the functional track or 4 for the AI track). It is run in parallel with the previous steps.

After the preprocessing procedure is finished, we spawn N workers for patch validation, two of which are spawned after the patch generation step is finished to avoid overloading the system. Each worker repeatedly fetches a candidate patch from the patch dispatcher thread, compiles and tests the patch, and marks it as plausible if the test succeeds.

3.5 Epilogue

When all patches are validated, or the time reaches 30 seconds before the timeout, the main script immediately saves plausible patches onto the disk and then terminates.

In case there are more than five plausible patches, which exceeds the limit of the competition, ET picks five plausible patches that maximize the number of covered locations. For example, when all plausible patches come from four suspicious locations, it picks two patches for the first location, and one patch for each of the other three locations.

4 RESULTS

ET is ranked as #1 in the *Functional Errors - Java* track and #2 in the *AI Generated Code - Java* track. Therefore, it has shown the ability to fix bugs in open-source projects like JFreeChart (as shown in Figure 2) and code generated by large language models.

After inspecting the results, we have found one implementation bug that affects the performance of ET. The current patch validation process only considers classes named as `Test*`, `*Test` or `*Tests` as test cases, following the convention of Maven. However, the benchmark has several test cases that fall beyond this pattern (e.g., `new_Test_115689`) and are thus ignored by ET. We are unable to fix this issue because the competition does not allow modifying the tool after the submission deadline. We expect a performance increase if this issue is fixed.

5 CONCLUSION

In this paper, we proposed ET, an end-to-end Java program repair tool for Java. In the APR-Comp 2024, ET is ranked as #1 in the *Functional Errors - Java* track and #2 in the *AI Generated Code - Java* track, showing the ability to repair real-world bugs.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant No. 2022YFB4501902, and ZTE Industry-University-Institute Cooperation Funds under Grant No.HC-CN-20210319008.

REFERENCES

[1] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan JC Van Gemund. 2009. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software* 82, 11 (2009), 1780–1792.
[2] Apache Software Foundation. 2023. *Apache Maven.* https://maven.apache.org/
[3] Apache Software Foundation. 2023. *mvnd - the Maven Daemon.* https://github.com/apache/maven-mvnd
[4] ASSERT-KTH. 2023. *flacoco.* https://github.com/ASSERT-KTH/flacoco
[5] ZeroMQ authors. 2023. *ZeroMQ.* https://zeromq.org/
[6] NUS-APR. 2023. *Cerberus Framework.* https://github.com/nus-apr/cerberus
[7] Python Software Foundation. 2023. *difflib — Helpers for computing deltas.* https://docs.python.org/3/library/difflib.html#difflib.unified_diff
[8] Ridwan Shariffdeen. 2023. *1st International Competition on Automated Program Repair.* https://apr-comp.github.io/
[9] StarCoder. 2023. *StarCoder.* https://huggingface.co/bigcode/starcoder
[10] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2022. *An Empirical Investigation into Learning Bug- Fixing Patches in the Wild via Neural Machine Translation.* https://doi.org/10.5281/zenodo.7478730
[11] Yuan-An Xiao. 2023. *Docker image xmcp/et:231004.2.* https://hub.docker.com/layers/xmcp/et/231004.2/images/sha256-78351cfb9bafad82d61bc594719f22f4f596ad52da669bc98bd856ef640e7c27
[12] Yuan-An Xiao. 2023. *Pull request of ET.* https://github.com/nus-apr/cerberus/pull/115
[13] Yuan-An Xiao, Chenyang Yang, Bo Wang, and Yingfei Xiong. 2023. Accelerating Patch Validation for Program Repair with Interception-Based Execution Scheduling. arXiv:2305.03955 [cs.SE] https://doi.org/10.48550/arXiv.2305.03955
[14] Yuan-An Xiao, Chenyang Yang, Bo Wang, and Yingfei Xiong. 2023. ExpressAPR: Efficient Patch Validation for Java Automated Program Repair Systems. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE).* 2038–2041. https://doi.org/10.1109/ASE56229.2023.00012
[15] Qihao Zhu Yuan-An Xiao. 2023. *ET repair tool.* https://zenodo.org/doi/10.5281/zenodo.8405059
[16] Qihao Zhu, Zeyu Sun, Yuan-an Xiao, Wenjie Zhang, Kang Yuan, Yingfei Xiong, and Lu Zhang. 2021. A Syntax-Guided Edit Decoder for Neural Program Repair. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 341–353. https://doi.org/10.1145/3468264.3468544
[17] Qihao Zhu, Zeyu Sun, Wenjie Zhang, Yingfei Xiong, and Lu Zhang. 2023. Tare: Type-Aware Neural Program Repair. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1443–1455.