EnCus: Customizing Search Space for Automated Program Repair

Seongbin Kim*, Sechang Jang*, Jindae Kim†, and Jaechang Nam*‡
*Handong Global University, Pohang, South Korea
†Seoul National University of Science and Technology, Seoul, South Korea
{sbkim, scjang}@handong.edu, jindae.kim@seoultech.ac.kr, jcnam@handong.edu

***Abstract*—The primary challenge faced by Automated Program Repair (APR) techniques in fixing buggy programs is the search space problem. To generate a patch, APR techniques must address three critical decisions: where to fix (location), how to fix (operation), and what to fix with (ingredient). In this study, we propose `EnCus`, a novel approach that customizes the search space of ingredients and mutation operators during patch generation. `EnCus` acts as an APR wingman, using an ensemble-based strategy to customize the search space. The search space is customized by extracting edit operations that are used to fix similar bug-introducing changes from existing patches. `EnCus` applies an ensemble of edit operations extracted from three open source project pools and three Abstract Syntax Tree (AST)-level code differencing tools. This ensemble provides complementary perspectives on the buggy context. To evaluate this approach, we integrate `EnCus` to an existing context-based APR tool, ConFix. Using `EnCus`, the extensive search space of ConFix is reduced to ten recommended patches. `EnCus` was evaluated on single-line Defects4J bugs, successfully generating 20 correct patches which performs comparably to state-of-the-art context-based APR techniques.**

***Index Terms*—automatic program repair, search space, code differencing**

I. Introduction

Automated Program Repair (APR), as noted by Long and Rinard [1], faces the challenge of effectively navigating the search space. To fix buggy source code, APR techniques must make three critical decisions: where to fix (location), how to fix (operation), and what to fix with (ingredient) [2]. APR techniques can generate a patch for a repair only when they make the right decisions for the three areas. To investigate and determine the final solution (patch) of a buggy code, APR techniques need to consider possible combinations of the three decisions, namely, the search space. Consequently, various strategies on these factors can determine a unique search space for each APR technique.

Many APR techniques employ heuristics-based approaches that mainly focus on creating a large search space to generate a fixing patch and then navigating it using search heuristics to find the patch efficiently [3]. To create the search space, early APR techniques [4]–[6] used a handful of different types of edits, which have not been enough to contain correct patches for bugs. To expand the search space, later approaches [7]–[13] leveraged various information collected from history, such as human-written bug fixes.

However, the trade-off between a large search space with more patches and the difficulty of finding the correct patches still exists. As the search space expands, navigation becomes increasingly complex. Without an effective navigation strategy, expanding the search space does not yield meaningful results.

To address the challenges of heuristic-based APR techniques, we propose `EnCus`, a novel **En**semble-based search space **Cus**tomization for APR techniques. A previous approach, SPI [14], demonstrated the potential of reducing the search space for context-based APR tools. Building on this foundation, our approach takes advantage of the reduced search space to implement an ensemble-based method that customizes the search space by combining diverse perspectives. Specifically, `EnCus` combines an ensemble of three open-source project pools and three AST-level code differencing tools to provide complementary views of the bug context. This synergy between the customized project pools and the AST-level differencing tools enhances `EnCus`'s ability to navigate the search space efficiently and identify correct patches.

Due to the challenges associated with search spaces, recent studies in APR have increasingly focused on deep learning-based approaches, which have demonstrated superior performance compared to heuristic-based methods [15]. When problems related to search spaces are effectively addressed, heuristic-based approaches can offer significant potential. Recent deep learning-based APR approaches have achieved improved results when integrated with concepts derived from heuristic-based methods [15]–[17]. In this context, `EnCus` provides valuable insights.

We evaluated `EnCus` on 125 single-location code modifications from the Defects4J [18] dataset. Building on SPI, `EnCus` further customizes the reduced search space using its ensemble approach, ultimately fixing 20 bugs in total, including 8 bugs that ConFix could not address. We share the replication package.[1]

The contributions of our study are as follows.

- **Search Space Customization**: We introduce `EnCus`, a novel approach designed to customize the search space for APR tools by using an ensemble-based method that integrates three project pools and three AST-level code differencing tools.

‡ Corresponding author

[1] https://github.com/HandongSF/EnCus

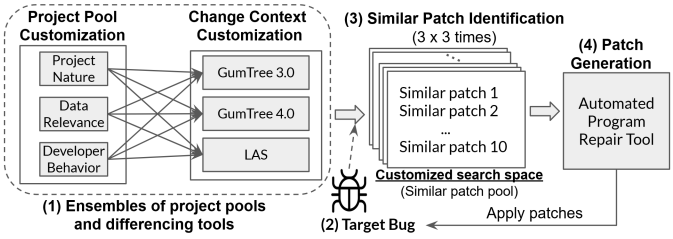

Fig. 1. Overview of EnCus.

- **Evaluation and Performance Improvement**: We demonstrate that integrating EnCus into an existing APR tool improves its performance.

II. Background

The main concern of heuristics-based automated program repair is the size of the search space. Jang et al. proposed SPI [14], which reduces the search space during the construction of a mutation operation pool. SPI collects Bug-Introducing Changes (BIC) and their corresponding Bug-Fixing Commits (BFC) from open-source projects. SPI identifies BICs with similar contexts to the target bug based on contextual similarity based on a tree-based code differencing tool. Then, it recommends BFCs corresponding to the identified BIC, creating a customized search space for the APR tool. SPI used only 0.15% bug-fixing commits compared to ConFix and showed a comparable result [9], [14].

The effectiveness of SPI relies heavily on the quality of data used for pool construction. If the collected data lacks diversity or relevance to the target projects, the resulting pool may fail to generalize to new bug contexts.

To enhance effectiveness, the pool should prioritize actively maintained projects that reflect current bug-fixing practices, exclude irrelevant changes, and focus on projects with features similar to the target, ensuring the patches are relevant and applicable [9], [13], [19].

The process of identifying similar bug contexts is strongly influenced by the choice of the code differencing tool. Identifying similar bug contexts plays a pivotal role in SPI as it establishes the foundation for recommending appropriate bug-fixing commits. However, code differencing tools often produce varying results due to their distinct algorithms [20], [21]. GumTree [22], the most widely used tool in this domain, focuses on generating fine-grained edit scripts that closely reflect human edits. Its latest iteration, GumTree4.0 [23] improves on scalability and efficiency for larger codebases. Additionally, LAS [24] introduces a location-aware approach to improve accuracy by focusing on multiple-node matching.

III. Approach

A. Overview

As shown in Fig. 1, EnCus constructs ensembles of project pools and AST-level code differencing tools to guide the repair process. It begins by using three open-source project pools according to project nature, data relevance, and developer behavior to identify contexts similar to the buggy code. The contexts of these changes are computed using an ensemble of three differencing tools. Constructing the ensembles is a one-time process, and they are utilized to define a customized search space that consists of just 90 similar patches, 10 patches for each pair of a pool and a change context, to fix the target bug. Once similar patches are identified, they are supplied to an APR tool to generate the patch for the bug.

B. Project Pool Customization

EnCus mines open-source projects to create a customized pool of bug-fixing contexts. This pool is populated with contexts of Bug-Introducing Changes (BIC) extracted from the historical changes in open-source projects. EnCus employs an ensemble approach, constructing three distinct pools based on the following characteristics:

- Project nature: Projects with similar functionalities to the target project were included, based on the assumption that similar functionality leads to similar bug-fixing patterns.
- Data relevance: A refined pool was created by filtering for changes directly impacting buggy code while excluding unrelated edits, such as format or documentation changes.
- Developer behavior: Actively maintained and frequently updated projects were selected, ensuring the pool represents up-to-date and practical bug-fixing practices.

C. Change Context Customization

After constructing the project pools, EnCus identifies past bugs with contexts similar to the target bug. EnCus uses an ensemble of three AST-level code differencing tools to identify the contexts. This ensemble approach focuses on describing the same change with different edit operations and contexts, providing a variety of potential operations to fix the bug.

Relying on a single differencing tool could lead to missing applicable changes due to minor differences between the buggy context and the previously collected operations, even if the necessary ingredients are present. By leveraging multiple tools, EnCus mitigates this issue, ensuring that relevant contexts and edit operations are not overlooked.

D. Patch Generation

Once the top similar Bug-Introducing Changes (BIC) are identified based on their similarity of AST-level code differences, the paired Bug-Fixing Commits (BFC) are provided to ConFix. These selected BFCs represent a curated set of bug-fixing edits. By leveraging this customized search space (similar patch pool) of bug fixing edits, the search space for potential patch candidates is narrowed, enabling the repair process to focus on promising solutions rather than exhaustively exploring the entire space of possible edits.

IV. Experimental Setup

We address the following research questions:

- **RQ1:** How effective is EnCus in improving the performance of an APR tool?
- **RQ2:** What is the impact of the ensemble approaches on the performance of EnCus?

A. Evaluation

To evaluate EnCus, we used Defects4J [18] version 2.0.1 as the benchmark dataset. Experiments were conducted on projects Chart, Closure, Lang, Math, and Time, which previous APR tools [8], [9], [12], [13] attempted to repair. Deprecated bugs in the older versions were not counted toward the results.

The evaluation was restricted to single-location code modifications to align with the capabilities of ConFix in these experiments. Furthermore, EnCus generates candidate bug-fixing patches at the modification level of a single source code file. Consequently, bugs requiring patches across multiple files or involving multiple locations within a single file were excluded, resulting in a total of 125 bugs for evaluation.

To evaluate the impact of EnCus on the performance of APR tools, the actual bug location information was provided.

B. Open-Source Project Pools

The three project pools consist of the following projects:

- Project nature: We used an online resource, Awesome Java [25], which curates libraries with similar functionalities. For each project in Defects4J [18], we selected the top two related projects: Closure was matched with ANTLR [26] and JFlex [27], Math with MALLET [28] and Smile [29], Chart with XChart [30] and BioJava [31], Lang with Guava [32] and commons-Text [33], and Time with ical4j [34] and Time4J [35].
- Data relevance: We used GrowingBugRepository [19], a repository of 250 projects, that contains patches with only changes relevant to the bug. To match our target, only single-line bugs were extracted.
- Developer behavior: Using the bug-tracking software Jira [36], we selected five Apache projects known for their frequent bug-fixing activity: Beam [37], Cassandra [38], Hadoop [39], jUDDI [40], and Spark [41].

C. AST-Level Code Differencing Tools

We evaluated EnCus using three AST-level code differencing tools. GumTree 3.0 [22], GumTree 4.0 [23], and LAS [24]. These tools have distinct approaches to extracting edit scripts, resulting in different change contexts.

D. APR Tools

For the APR tool integrated with EnCus, we selected ConFix [9]. To evaluate the potential of the approach, ConFix was executed using a single patching strategy: *FLFreq & Hash Match*. Since EnCus significantly reduces the search space, we set a time limit of 30 minutes per bug for patch generation. If no patch was generated within this time frame, it was considered a failure. The accuracy of the patches was assessed by manually comparing the generated patches with the ground-truth solutions.

We evaluated the performance of EnCus against four other context-based APR tools to provide a broader comparison: ssFix [13], CapGen [12], SimFix [8], and ConFix [9]. These tools were selected for their heuristic-based approaches that leverage bug context, external patch context, or both, making them relevant to our customized search-space reduction approach. The results were obtained from each tool's public artifacts [42]–[45].

V. Results

A. RQ1: How effective is EnCus in improving the performance of an APR tool?

To evaluate the impact of integrating EnCus with an APR tool, we compared the performance of EnCus + ConFix, SPI + ConFix, standalone ConFix [9], ssFix [13], CapGen [12], and SimFix [8] on single-line bugs in Defects4J [18]. Table I summarizes the results.

TABLE I
PERFORMANCE COMPARISON OF ENCUS + CONFIX VS. SPI + CONFIX VS. CONTEXT-BASED APR TOOLS.

Projects	EnCus + ConFix	SPI + ConFix	ConFix	ssFix	CapGen	SimFix
Chart	3	3	4	2	4	2
Closure	6	3	4	1	N/A	4
Lang	3	2	5	5	4	3
Math	8	4	6	5	11	7
Time	0	0	1	0	0	0
Sum	20	12	20	13	19	16

The results demonstrate that EnCus + ConFix outperforms SPI + ConFix and performs equivalently compared to ConFix in terms of the total number of bugs repaired, successfully addressing 20 bugs compared to 12 by SPI + ConFix and 20 by standalone ConFix. This highlights the improvement in the effectiveness EnCus's ensemble-based approach provides, by succeeding in generating a comparable number of correct patches. In addition, EnCus surpasses several baseline tools, including ssFix (13), CapGen (19), and SimFix (16).

EnCus introduced eight new bug fixes that ConFix was unable to generate: Closure-10, 70, 86, 133, Lang-59, Math-57, 58, 59. EnCus + ConFix compared to SPI + ConFix, which repaired 12 bugs, generated nine new bug fixes: Closure-70, 73, 133, Lang-57, Math-33, 34, 57, 58, and Chart-24, demonstrating the effectiveness of its ensemble-based customization of the search space.

For example, Closure-70 was repaired using combinations of either GumTree 4.0 or LAS. In contrast, SPI used only GumTree 3.0 to identify contexts and failed to generate a correct patch. Math-57 was repaired using the data relevance Pool. The effect of focusing on the core edits allowed EnCus to customize a search space for ConFix to generate a correct fix that ConFix's search space originally overlooked.

However, the number of correct patches generated by EnCus + ConFix did not exceed the total number of fixes made by ConFix. Several reasons contributed to this limitation. For cases Chart-1, Closure-38, 92, Lang-24, and Time-19, EnCus + ConFix generated plausible patches earlier than generating correct patches, which preemptively terminated the patch generation process. In the case of Lang-26, 51, and Math-75, EnCus had difficulties collecting Bug Introducing Changes (BIC) using the Defects4j [18] framework.

EnCus requires information from before the bug was introduced; however, for Lang-26 and Math-75, this information could not be retrieved during the code revision checkout process. Lang-51 faced inconsistencies between Defects4J's metadata and the actual repository data, leading to mismatches during the checkout process.

Also, EnCus + ConFix failed to repair a bug SPI + ConFix successfully fixed, Chart-1. The human patch for Chart-1 involves a simple change from != to == in a conditional statement. While EnCus successfully identifies the change context to be a boolean operator modification, it fails to identify the concrete change from != to ==. This is because EnCus uses AST-level code differencing, while SPI applies another stage of text-based differencing to identify concrete syntax change contexts.

B. RQ2: What is the impact of the ensemble approaches on the performance of EnCus?

TABLE II
NUMBER OF CORRECT FIXES GENERATED BY THE ENSEMBLE APPROACH

	GumTree 3.0	GumTree 4.0	LAS
Project Nature	8	12	12
Data Relevance	15	14	15
Developer Behavior	12	14 (1)	10 (1)

Table II shows the number of correct fixes generated with different customization strategies. The parenthesized numbers represent a unique fix that only a specific combination of the customization strategies generated. The developer behavior pool contributed the most with two unique patches.

The pool collected based on data relevance shows minimal influence from code differencing tools. This is because code differencing tools tend to exhibit varying perspectives, especially for long modifications. The pool, curated with a focus on data relevance, excludes unrelated modifications and primarily consists of short edits. In practice, all bug fixes in this pool were identical, where only one case (Closure-73) failed to be fixed using GumTree 4.0 [23].

In contrast, the pool collected based on developer behavior exhibits strong synergy with GumTree 4.0. GumTree 4.0 is designed to be scalable for large codebases, and the developer behavior pool, collected to focus on active bug fixing, includes substantial modifications within large code repositories compared to the others.

Based on the Pool Customization Strategy and Code Differencing Tool Customization Strategy, the pool based on developer behavior produced two fixes that the other pool could not generate, respectively. Also, each code differencing tool produced a fix that the different tools could not generate. The ensemble-based approach in EnCus demonstrates advantages by combining pools and differencing tools, broadening the search space while maintaining precision in patch generation.

VI. DISCUSSION

EnCus demonstrates scalability by leveraging its customizable search space approach. Currently, it operates with three pools and three differencing tools. However, its design allows for the addition of more pools and tools. This flexibility enables EnCus to scale as new data sources or advanced differencing tools become available. Once the ensemble is formed, the similar patch pool is significantly reduced. By narrowing the search space, EnCus can reduce computational overhead, allowing for multiple iterations and better results.

The eight additional bug fixes generated by EnCus, compared to ConFix [9], demonstrate the effectiveness of customized search spaces. With more advanced methods for mining change pools and extracting AST-level changes, EnCus has the potential to achieve better performances.

EnCus's performance was evaluated using heuristic-based APR tools, focusing on generating an efficient pool of mutation operations in the preliminary stages. Recent trends in APR techniques predominantly use deep learning, which has demonstrated superior performance compared to heuristic-based methods [46]. By leveraging EnCus's ability to analyze and compare code changes, it can enhance template-based and neural repair methods by refining template selection and assisting in generating higher-quality patches [15]–[17].

Defects4J [18] is the benchmark used to evaluate the performance of EnCus. To reproduce each bug, Defects4J injects the bug into a project where the bug has already been fixed. According to EnCus's approach, the BIC is the modification in the code that transitions from the fixed code to the buggy code. To evaluate EnCus's performance, Defects4J needed to be used as the benchmark.

EnCus has the potential of over-fitting in the constructed pools of bug-fixing contexts. Since the pools were curated based on specific criteria, there was a risk that the diversity of the included bug-fixing patterns might not be sufficiently generalized to new contexts. Additionally, the use of an ensemble of AST-level code differencing tools can overemphasize certain structural similarities, potentially ignoring semantic information.

VII. CONCLUSION

This study introduces EnCus, a novel approach to reduce the search space in APR. By customizing the search space of patch ingredients and mutation operators, EnCus leverages an ensemble of edit operations extracted from diverse open-source project pools and AST-level code differencing tools. EnCus not only narrows the extensive search space but also enhances the efficiency and accuracy of heuristic-based APR tool while reducing the computational burden. We are expecting EnCus can serve as a foundational tool not only for improving heuristic-based methods but also for enhancing learning-based APR approaches.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2024-00457866).

REFERENCES

[1] F. Long and M. Rinard, "An analysis of the search spaces for generate and validate patch generation systems," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 702–713.

[2] M. Wen, Y. Liu, and S.-C. Cheung, "Boosting automated program repair with bug-inducing commits," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 77–80.

[3] C.-P. Wong, P. Santiesteban, C. Kästner, and C. Le Goues, "Varfix: balancing edit expressiveness and search effectiveness in automated program repair," in *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2021, pp. 354–366.

[4] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "Genprog: A generic method for automatic software repair," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 54–72, 2012.

[5] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic patch generation learned from human-written patches," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 802–811.

[6] F. Long and M. Rinard, "Staged program repair with condition synthesis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 166–178.

[7] J. Bader, A. Scott, M. Pradel, and S. Chandra, "Getafix: Learning to fix bugs automatically," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–27, 2019.

[8] J. Jiang, Y. Xiong, H. Zhang, Q. Gao, and X. Chen, "Shaping program repair space with existing patches and similar code," in *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis*, 2018, pp. 298–309.

[9] J. Kim and S. Kim, "Automatic patch generation with context-based change application," *Empirical Software Engineering*, vol. 24, pp. 4071–4106, 2019.

[10] K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandé, "Tbar: Revisiting template-based automated program repair," in *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, 2019, pp. 31–42.

[11] F. Long, P. Amidon, and M. Rinard, "Automatic inference of code transforms for patch generation," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 727–739.

[12] M. Wen, J. Chen, R. Wu, D. Hao, and S.-C. Cheung, "Context-aware patch generation for better automated program repair," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 1–11.

[13] Q. Xin and S. P. Reiss, "Leveraging syntax-related code for automated program repair," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 660–670.

[14] S. Jang, J. Choi, S. Kim, J. Kim, and J. Nam, "Spi: Similar patch identifier for automated program repair," in *Korea Computer Congress 2024 (KCC)*. KIISE, 2024, pp. 354–356.

[15] X. Meng, X. Wang, H. Zhang, H. Sun, X. Liu, and C. Hu, "Template-based neural program repair," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1456–1468.

[16] K. Huang, J. Zhang, X. Meng, and Y. Liu, "Template-guided program repair in the era of large language models," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2024, pp. 367–379.

[17] C. Liu, P. Cetin, Y. Patodia, B. Ray, S. Chakraborty, and Y. Ding, "Automated code editing with search-generate-modify," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, pp. 398–399.

[18] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 international symposium on software testing and analysis*, 2014, pp. 437–440.

[19] Y. Jiang, H. Liu, N. Niu, L. Zhang, and Y. Hu, "Extracting concise bug-fixing patches from human-written patches in version control systems," in *IEEE/ACM 43rd International Conference on Software Engineering (ICSE 2021)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 686–698. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE43902.2021.00069

[20] Y. Fan, X. Xia, D. Lo, A. E. Hassan, Y. Wang, and S. Li, "A differential testing approach for evaluating abstract syntax tree mapping algorithms," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1174–1185.

[21] P. Alikhanifard and N. Tsantalis, "A novel refactoring and semantic aware abstract syntax tree differencing tool and a benchmark for evaluating the accuracy of diff tools," *ACM Transactions on Software Engineering and Methodology*, 2024.

[22] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and accurate source code differencing," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 313–324.

[23] J.-R. Falleri and M. Martinez, "Fine-grained, accurate and scalable source differencing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[24] J. Kim and S. Kim, "Location aware source code differencing for mining changes," Tech. rep., Hong Kong University of Science and Technology. https://github …, Tech. Rep., 2016.

[25] akullpp, "Awesome java," https://github.com/akullpp/awesome-java, 2024.

[26] antlr, "Antlr," https://github.com/antlr/antlr4, 2024.

[27] jflex de, "Jflex," https://github.com/jflex-de/jflex, 2023.

[28] mimno, "Mallet," https://github.com/mimno/Mallet, 2024.

[29] haifengl, "Smile," https://github.com/haifengl/smile, 2024.

[30] knowm, "Xchart," https://github.com/knowm/XChart, 2024.

[31] biojava, "biojava," https://github.com/biojava/biojava, 2024.

[32] google, "guava," https://github.com/google/guava, 2024.

[33] apache, "commons-text," https://github.com/apache/commons-text, 2024.

[34] ical4j, "ical4j," https://github.com/ical4j/ical4j, 2024.

[35] MenoData, "Time4j," https://github.com/MenoData/Time4J.git, 2024.

[36] Atlassian, "Jira," https://www.atlassian.com/software/jira, 2024.

[37] apache, "beam," https://github.com/apache/beam, 2024.

[38] ——, "cassandra," https://github.com/apache/cassandra, 2024.

[39] ——, "hadoop," https://github.com/apache/hadoop, 2024.

[40] ——, "juddi," https://github.com/apache/juddi, 2023.

[41] ——, "spark," https://github.com/apache/spark, 2024.

[42] qixin, "ssfix," https://github.com/qixin5/ssFix/tree/master/expt0/patch/ssFix, 2020.

[43] MingWEN-CS, "Capgen," https://github.com/MingWEN-CS/CapGen/tree/master/Patches, 2020.

[44] xgdsmileboy, "Simfix," https://github.com/xgdsmileboy/SimFix/tree/master/final/result, 2023.

[45] thwak, "confix2019result," https://github.com/thwak/confix2019result/tree/master/patches, 2019.

[46] Q. Zhang, C. Fang, Y. Ma, W. Sun, and Z. Chen, "A survey of learning-based automated program repair," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 2, pp. 1–69, 2023.