CLEAR: Contextual Learning for Effective Automated Repair of OD-Flaky Tests

Eric Wang*
ericrwang2011@utexas.edu
University of Texas at Austin
Austin, Texas, USA

Abstract

Flaky tests, which produce non-deterministic outcomes across repeated executions, are a pervasive challenge in modern software development. Among them, order-dependent (OD) flaky tests—those whose behavior is influenced by the order of execution or shared state—are challenging to diagnose and repair. These tests disrupt continuous integration pipelines, obscure defects, and erode developer trust in automated testing. Existing approaches to flaky test repair often rely on extensive project-level context or manual intervention, limiting their scalability and applicability.

This paper presents an LLM-driven framework for repairing OD flaky tests, providing access only to the test code. By leveraging GPT-based large language models, our method analyzes the test code in isolation to propose targeted fixes for common OD patterns such as shared state dependencies, non-deterministic behavior, and timing issues. To improve repair accuracy, we combine a test bank of real-world and generated flaky test examples with dynamic similarity-based example selection, providing the LLM with highly relevant in-context learning data.

Furthermore, we evaluate how increasing the number of examples provided for in-context learning affects the repair results of flaky tests. This analysis aims to determine whether providing more code examples improves the repair accuracy and generalization of the framework. By addressing this critical subset of flaky tests with minimal input requirements, our approach offers a scalable and automated solution for enhancing test reliability and strengthening software development pipelines.

Keywords: Flaky Tests, Large Language Models, In-Context Learning , Software Testing

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Eric Wang. 2018. CLEAR: Contextual Learning for Effective Automated Repair of OD-Flaky Tests. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Flaky tests, which exhibit inconsistent outcomes across repeated executions, are a pervasive challenge in modern software development. Such tests undermine the reliability of testing pipelines, leading to wasted developer time, obscured defects, and delayed software releases. Among the various types of flaky tests, order-dependent (OD) flaky tests—those whose behavior is influenced by the order of execution or shared states—are particularly problematic. These tests often fail or pass depending on the sequence in which they or other tests are executed, causing significant challenges in test maintenance and debugging. The scale of this issue is highlighted by reports from major technology companies. For example, a report from Google shows that they had 1.6M test failures on average each day, and 73K of them were caused by flaky tests [2] [8]. Several other companies like Microsoft, Facebook [1], Netflix, Mozilla, and Salesforce have reported similar experiences, where flaky tests have significantly delayed their deployment process [7] [9] [11]. These challenges are further compounded in the case of OD flaky tests, which often require extensive manual effort to isolate and repair due to their dependence on shared states and execution orders [5].

While extensive research has explored detecting and categorizing OD flaky tests, much of this work has focused on Java projects or relied on runtime dependencies such as polluters and state-setters [4] [6] [7] [8] [10]. Less attention has been given to Python OD tests and methods to repair these tests without requiring access to the entire project context [12] [3]. Addressing this gap, we propose an LLM-driven framework for repairing OD flaky tests that analyzes only the test code itself. By isolating flaky test repair from project-level information, our approach provides a lightweight and scalable solution that reduces the cost and complexity of repairs.

Central to our method is the use of in-context learning with large language models (LLMs), such as ChatGPT, Gemini, LLaMA 3, and Claude. We enhance the repair process by dynamically selecting relevant examples from a combined test bank of real-world flaky tests and generated examples, leveraging semantic similarity to guide the LLM's repairs. This integration allows us to systematically address OD test patterns, such as shared state dependencies and non-deterministic execution order, even in constrained scenarios. A unique aspect of our framework is its iterative nature: validated fixes are added back into the test bank, effectively growing the repository of knowledge over time. This iterative enrichment not only improves the quality and diversity of the examples available for in-context learning but also ensures that the system continuously evolves to address new and emerging patterns of flaky tests. By enabling the test bank to expand dynamically, our method provides a scalable and ever-improving resource for flaky test repair.

To validate our framework, we evaluate the effectiveness of in-context learning starting with ChatGPT 3.5 Turbo, comparing their ability to repair OD flaky tests using a shared test bank. This investigation reveals not only the general benefits of optimized in-context learning but also highlights the relative performance of different models in handling these challenges. By focusing on this critical subset of flaky tests, our work provides an innovative and practical solution for improving test reliability and accelerating software development workflows.

Overall, this paper makes the following main contributions:

1. **CLEAR Framework**: A publicly available framework that streamlines the fixing of Python OD tests utilizing a shared test bank.
2. **Dataset**: A dataset of OD-related tests and applied patches for Python OD flaky tests. This dataset currently includes only the OD flaky tests and the corresponding repairs but is designed to expand and encompass additional flaky test categories.
3. **Study**: We perform a study on the effectiveness of in-context learning across different LLM models and demonstrate the benefits of providing optimized examples on repair effectiveness.

2 Background

Flaky tests are a common issue in software testing, characterized by non-deterministic behavior where a test may pass or fail without any changes to the code under test. Among the various types of flaky tests, order-dependent (OD) flaky tests are particularly challenging to address. These tests produce inconsistent results due to their reliance on the sequence in which tests are executed or their dependence on shared states that are not properly isolated. OD flaky tests often arise when:

- **Shared State Dependencies**: Tests inadvertently rely on or modify shared global variables, environment variables, or resources.
- **Execution Order Sensitivity**: The order in which tests are run affects the state of shared resources, leading to different outcomes.
- **Data Mutability**: Tests process unordered or mutable data without ensuring consistency in how it is accessed or modified.

These characteristics make OD flaky tests especially difficult to diagnose and repair, as they often require isolating complex interactions between tests. Addressing these challenges is critical for maintaining reliable test suites, especially in modern continuous integration (CI) environments where test flakiness can disrupt development workflows.

2.1 OD Test Example

The following test is flaky because the `test_state_change` function modifies the shared state variable `status.state` without resetting it. The `test_object` function relies on the value of `status.state`, which can vary depending on the execution order of the tests.

```python
from ..event import status

class TestEvent(object):
    def test_object(self):
        assert id(status.state) == id(status.state)

    def test_state_change(self):
        status.state = False
        assert status.state is False
```

Listing 1. OD flaky test example from open source project

The `test_object` function assumes that `status.state` is in its default state. However, if `test_state_change` executes before `test_object`, it changes the value of `status.state`, causing `test_object` to fail. This behavior makes the test order-dependent and flaky.

To address the flakiness, `test_object` must explicitly reset `status.state` to a known value before performing its assertions. The fixed test code is shown below:

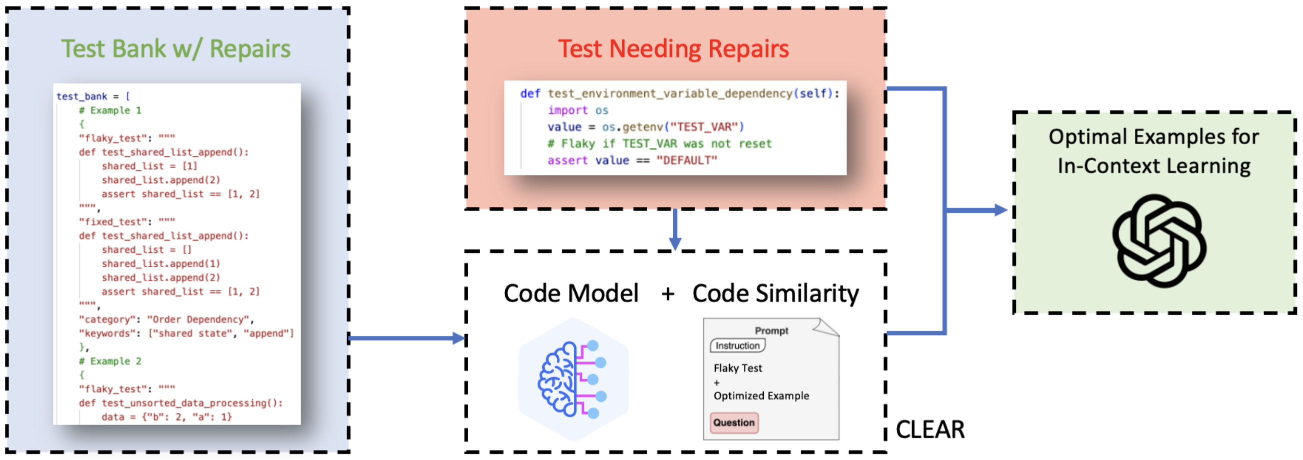
```python
from ..event import status

class TestEvent(object):
    def test_object(self):
        status.state = False # Reset state
        assert id(status.state) == id(status.state)

    def test_state_change(self):
        ...
```

Listing 2. OD flaky test example repaired

The fix ensures that `test_object` explicitly resets `status.state` to a clean state before performing its assertions. This eliminates the dependency on the execution order



```
test_bank = [
    # Example 1
    {
    "flaky_test": """
    def test_shared_list_append():
        shared_list = [1]
        shared_list.append(2)
        assert shared_list == [1, 2]
    """,
    "fixed_test": """
    def test_shared_list_append():
        shared_list = []
        shared_list.append(1)
        shared_list.append(2)
        assert shared_list == [1, 2]
    """,
    "category": "Order Dependency",
    "keywords": ["shared state", "append"]
    },
    # Example 2
    {
    "flaky_test": """
    def test_unsorted_data_processing():
        data = {"b": 2, "a": 1}
```

```
def test_environment_variable_dependency(self):
    import os
    value = os.getenv("TEST_VAR")
    # Flaky if TEST_VAR was not reset
    assert value == "DEFAULT"
```

Figure 1. Overview of CLEAR pipeline.

of other tests, thereby addressing the flaky behavior. By adhering to the principle of test isolation, the test suite becomes more reliable and robust.

3 CLEAR

The CLEAR framework repairs order-dependent flaky tests by leveraging a dynamically evolving test bank and LLM-driven in-context learning. The process involves:

1. **Test Bank Querying**: A semantic similarity search identifies the most relevant examples of flaky tests and their fixes from the test bank.
2. **Example Selection**: The top-ranked example(s) are selected as in-context learning examples for the LLM.
3. **Repair Generation**: The flaky test and selected examples are input to the LLM, which generates a repair.
4. **Test Bank Expansion**: Validated repairs are added to the test bank to enhance future performance.

3.1 CLEAR Test Bank Generation

The test bank was constructed using a combination of real-world data and synthetic examples. For real-world data, we extracted flaky tests and corresponding repair patches from the iPFlakies dataset [12]. Only tests with repairs directly applied to them were included in the dataset. Tests where the flaky behavior was caused by another function and the patch was applied to that function rather than the failing test itself, were excluded. In total, the test bank contains flaky tests from 15 open source projects sourced from iPFlakies dataset [12]. To complement the real-world examples, we generated synthetic flaky tests and repairs to cover common order-dependent patterns such as shared state dependencies, timing issues, and execution order sensitivity. The resulting test bank is a unified repository containing flaky tests, their fixes, and metadata such as issue type and context.

3.2 Optimized Example Selection

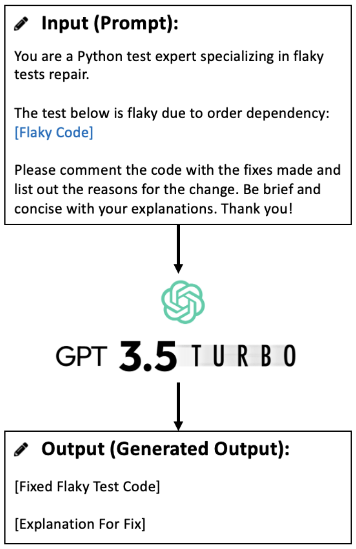
To determine the most relevant examples from the test bank, the CLEAR framework performs the following steps. First, the input flaky test is tokenized and converted into a vector embedding using a pre-trained model such as UniXcoder or CodeBERT, capturing its semantic and structural information. Each example in the test bank is similarly preprocessed and embedded into vector form. Cosine similarity is then calculated between the embedding of the input flaky test and all test bank examples to measure relevance:

$$\text{Cosine Similarity} = \frac{\text{Input Embedding} \cdot \text{Example Embedding}}{\|\text{Input Embedding}\| \|\text{Example Embedding}\|}.$$

Examples are ranked by similarity score, and the top-ranked example(s) are selected as in-context learning examples for the LLM. If multiple examples have similar scores, diversity in issue type (e.g., shared state or timing issues) is considered to improve generalization. The number of selected examples is limited by the LLM's input token capacity. This process ensures that the examples provided to the LLM are semantically aligned with the input flaky test, enabling accurate and context-aware repairs.

3.3 Repair Prompt Engineering

To repair order-dependent flaky tests using GPT, we designed two types of prompts, shown in Figure 2 and Figure 3. The first type (Figure 2) provides only the flaky test code and describes the issue as "order dependency," requesting the model to generate a fix along with comments explaining the changes. The second type (Figure 3) extends the first by including one or more examples of similar flaky tests and their fixes, dynamically selected from the test bank based on semantic similarity. For both types, the model is instructed to generate the repaired test code with comments highlighting the fixes and a brief explanation of the changes.



Input (Prompt):

You are a Python test expert specializing in flaky tests repair.

The test below is flaky due to order dependency:
[Flaky Code]

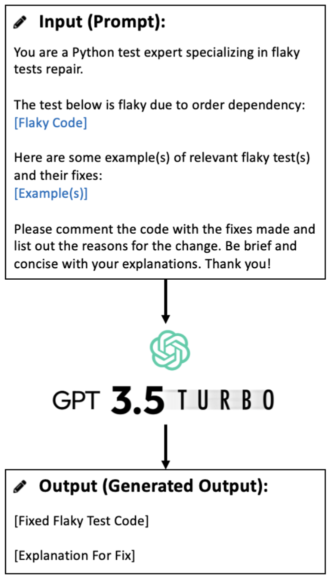
Please comment the code with the fixes made and list out the reasons for the change. Be brief and concise with your explanations. Thank you!

Output (Generated Output):

[Fixed Flaky Test Code]

[Explanation For Fix]

Figure 2. Flaky test fix using GPT-3.5 Turbo without relevant example(s) for in-context learning



Input (Prompt):

You are a Python test expert specializing in flaky tests repair.

The test below is flaky due to order dependency:
[Flaky Code]

Here are some example(s) of relevant flaky test(s) and their fixes:
[Example(s)]

Please comment the code with the fixes made and list out the reasons for the change. Be brief and concise with your explanations. Thank you!

Output (Generated Output):

[Fixed Flaky Test Code]

[Explanation For Fix]

Figure 3. Flaky test fix using GPT-3.5 Turbo with relevant example(s) for in-context learning

In the future, we plan to broaden the capabilities of our framework by expanding the test bank to include additional categories of flaky tests, such as non-deterministic behavior, concurrency issues, and external dependency flakiness. This will enhance the diversity of examples provided in prompts, improving the model's ability to generalize and repair a wider range of flaky test scenarios.

4 Evaluation

To evaluate the CLEAR framework, we evaluate the following RQs:

RQ1: Does the inclusion of dynamically selected examples improve the effectiveness of LLMs in repairing order-dependent flaky tests?
RQ2: Does increasing the number of provided examples impact the quality of the repair?

5 Methodology

To address RQ1, we use the iPFlakies dataset to identify flaky tests where the fixes are applied directly to the flaky test itself. Tests, where the flakiness is caused by another test and the fix, is applied elsewhere are excluded. We evaluate the LLM's performance using three prompt configurations: (1) the flaky test code without examples, (2) the flaky test code with one relevant example selected using cosine similarity, and (3) the flaky test code with the top three relevant examples. Repairs generated by the LLM are evaluated based on two criteria: whether the flakiness is resolved, the quality of the code, and how the repair compares to the ground-truth fix from iPFlakies.

For RQ2, we evaluate multiple LLMs (e.g., GPT, LLaMA, Claude) using flaky tests from the iPFlakies dataset. Each test is assessed under three prompt configurations like RQ1. The models are then compared based on the following metrics: (1) flakiness resolution by running the repaired test, (2) code quality by comparing repairs to the ground-truth fixes, and (3) runtime to assess efficiency. The goal is to evaluate the effectiveness and scalability of each model across example configurations.

6 Results

6.1 RQ1 Results

Table 1 shows that including examples improves the pass rate of repairs from 0% (no examples) to 40% (one or three examples).

0 Examples	1 Example	3 Examples
0%	40%	40%

Table 1. Pass Rates for Repairs Across Configurations

This demonstrates the utility of in-context learning for repairing order-dependent flaky tests. However, the success rate remains low due to several factors: (1) the complexity of OD flaky tests, which often involve intricate interactions with shared states or dependencies not captured in the test code alone, (2) constraints on input token capacity, limiting the number of examples provided to the LLM, and (3) variability in ground-truth fixes, where minor differences in implementation style may misalign with generated fixes.

6.2 RQ2 Results

Table 2 presents the Levenshtein edit distance between the generated repairs and the ground-truth fixes from the iPFlakies dataset.

0 Examples	**1 Example**	**3 Examples**
513	170	192

Table 2. Edit Distances for Repairs Across Configurations

The edit distance measures the number of token insertions, deletions, and substitutions required to transform the generated repair into the ground-truth fix. Including one example reduces the edit distance from 513 (no examples) to 170, indicating closer alignment with the ground truth. However, increasing to three examples results in a slight increase in edit distance to 192, suggesting diminishing returns. It is important to note that the iPFlakies dataset often contains one of many valid solutions for repairing flaky tests, and the generated fixes may differ in implementation style while still resolving the flakiness. These results highlight the importance of example relevance and the need for a balance between the number of examples and their contextual alignment with the input test.

7 Conclusion

This work presents CLEAR, a framework for repairing order-dependent flaky tests using large language models and in-context learning. By dynamically selecting relevant examples from a test bank, CLEAR offers a potential solution for addressing flaky test repairs with limited context. Our results indicate that providing examples does improve repair accuracy, though challenges such as low success rates and reliance on ground-truth solutions remain. While this study focuses on order-dependent flaky tests, the framework is designed to scale and adapt to other categories of flaky tests with future expansions of the test bank. CLEAR highlights the potential of combining LLMs with optimized prompting strategies for automated test repair and serves as a foundation for further research in this area.

References

[1] Maxime Cordy, Renaud Rwemalika, Mike Papadakis, and Mark Harman. 2019. FlakiMe: Laboratory-Controlled Test Flakiness Impact Assessment. A Case Study on Mutation Testing and Program Repair. *CoRR* abs/1912.03197 (2019). arXiv:1912.03197 http://arxiv.org/abs/1912.03197

[2] Emad Fallahzadeh and Peter C. Rigby. 2022. The impact of flaky tests on historical test prioritization on chrome. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice* (Pittsburgh, Pennsylvania) *(ICSE-SEIP '22)*. Association for Computing Machinery, New York, NY, USA, 273–282. https://doi.org/10.1145/3510457.3513038

[3] Sakina Fatima, Hadi Hemmati, and Lionel C. Briand. 2024. FlakyFix: Using Large Language Models for Predicting Flaky Test Fix Categories and Test Code Repair. *IEEE Transactions on Software Engineering* 50, 12 (Dec. 2024), 3146–3171. https://doi.org/10.1109/tse.2024.3472476

[4] Alex Gyori, August Shi, Farah Hariri, and Darko Marinov. 2015. Reliable testing: detecting state-polluting tests to prevent test dependency. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis* (Baltimore, MD, USA) *(ISSTA 2015)*. Association for Computing Machinery, New York, NY, USA, 223–233. https://doi.org/10.1145/2771783.2771793

[5] Sarra Habchi, Guillaume Haben, Mike Papadakis, Maxime Cordy, and Yves Le Traon. 2021. A Qualitative Study on the Sources, Impacts, and Mitigation Strategies of Flaky Tests. *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)* (2021), 244–255. https://api.semanticscholar.org/CorpusID:245005732

[6] Wing Lam, Stefan Winter, Angello Astorga, Victoria Stodden, and Darko Marinov. 2020. Understanding reproducibility and characteristics of flaky tests through test reruns in java projects. In *Proceedings - 2020 IEEE 31st International Symposium on Software Reliability Engineering, ISSRE 2020 (Proceedings - International Symposium on Software Reliability Engineering, ISSRE)*, Marco Vieira, Henrique Madeira, Nuno Antunes, and Zheng Zheng (Eds.). IEEE Computer Society, 403–413. https://doi.org/10.1109/ISSRE5003.2020.00045

[7] Wing Lam, Stefan Winter, Anjiang Wei, Tao Xie, Darko Marinov, and Jonathan Bell. 2020. A large-scale longitudinal study of flaky tests. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 202 (Nov. 2020), 29 pages. https://doi.org/10.1145/3428270

[8] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong, China) *(FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 643–653. https://doi.org/10.1145/2635868.2635920

[9] Valeria Pontillo, Fabio Palomba, and Filomena Ferrucci. 2024. Test Code Flakiness in Mobile Apps: The Developer's Perspective. *Information and Software Technology* 168 (2024), 107394. https://doi.org/10.1016/j.infsof.2023.107394

[10] August Shi, Wing Lam, Reed Oei, Tao Xie, and Darko Marinov. 2019. iFixFlakies: a framework for automatically fixing order-dependent flaky tests. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 545–555. https://doi.org/10.1145/3338906.3338925

[11] Amjed Tahir, Shawn Rasheed, Jens Dietrich, Negar Hashemi, and Lu Zhang. 2023. Test flakiness' causes, detection, impact and responses: A multivocal review. *Journal of Systems and Software* 206 (2023), 111837. https://doi.org/10.1016/j.jss.2023.111837

[12] Ruixin Wang, Yang Chen, and Wing Lam. 2022. iPFlakies: A Framework for Detecting and Fixing Python Order-Dependent Flaky Tests. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 120–124. https://doi.org/10.1145/3510454.3516846

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009