Faster Feedback with AI?
A Test Prioritization Study

Toni Mattis
Lukas Böhme
Eva Krebs
Martin Rinard
Robert Hirschfeld

Software Architecture Group
HPI, University of Potsdam, Germany

CSAIL
MIT, Cambridge, MA, USA

PAI/24 | Lund, Sweden | March 12
Motivation

Feedback during program activities help programmers to catch errors early.

Automated testing

- Running large test suite introduces *delays in feedback*
- Critical tests identifying a fault may *be hidden* between *many tests*
Regression Test Prioritization (RTP)

Objective: Rank all tests based on their relevance (e.g., run tests most likely to fail earlier)

Approaches of RTP subdivided by their input data:

- whole program vs. **program changes**
- historical data of previous test runs vs. **cold start**

Effective ideas

- Historical links between change and test
- **Semantic similarity** between (edited) code and test
Background

1. Semantic similarity: TF-IDF
2. Semantic similarity: Embeddings
3. Large Language Models
Background | Semantic Similarity

- Traditional approach: shared vocabulary via TF-IDF

```python
def login(user, password):
    if self.check(user, password):
        ...

def test_correct_password():
    user = self.example_user()
    ...
```

**Term frequency** (TF)

**Term importance**

*Inverse Document Frequency (IDF)*

Here: **Okapi BM25** (from Apache Lucene)
Background | Embeddings

- Vectors for (textual) data so that the **proximity** of two vectors measures the **semantic similarity** of their associated data

```python
def foo(): ...
def bar(): ...
def test_foo(): ...
def test_bar(): ...
```

\[
sim(a; b) = \cos(\alpha) = a \cdot b
\]
Background | **Language Models**

- Given a sequence of tokens *(prompt)*, computes probabilities for the *next token* out of all possible tokens
- Repeatedly append a probable token and re-run

```
def test_fib():
    assert fib(1) == 1
    assert fib(2) == 1
    assert fib(3) == 2
    assert fib(4) == 3
```
Approach | **AI-based Test Prioritization**

1. Data collection
2. LLM-based prioritization
3. Embedding-based prioritization
Approach | **Change-based Mutation Testing**

**Ground truth:** a test is **relevant** to a **change** when it **fails** if we **break** the change
### Change-based Mutation Testing

<table>
<thead>
<tr>
<th>Mutation</th>
<th>Change</th>
<th>Defect injected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>$1 + n$</td>
<td>$1 - n$</td>
</tr>
<tr>
<td></td>
<td>$a \times b$</td>
<td>$a / b$</td>
</tr>
<tr>
<td>Number</td>
<td><code>year = 2024</code></td>
<td><code>year = 42</code></td>
</tr>
<tr>
<td>String</td>
<td><code>prompt = &quot;Lund is awesome&quot;</code></td>
<td><code>prompt = &quot;&quot;</code></td>
</tr>
<tr>
<td>Condition</td>
<td><code>if conference.started:</code></td>
<td><code>if True:</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>if False:</code></td>
</tr>
</tbody>
</table>
Approach | **LLM-based RTP**

Instead of generating a test,

use the **probability** that the LLM **would have generated** a given test as test priority

- **Model:** *StableCode-3B*
LLM-based RTP | Representing Changes

- Include syntactically correct scope
- Comment out deleted code

```python
class AClass:
    def __init__(self):
        ...

    def method(self):
        # return self.value
        return self.value + 1
```

# /src/the_file.py

```
class AClass:
    def method(self):
        # return self.value
        return self.value + 1
```

# Test validating this change:

**Change**

**LLM Prompt**
**Approach | Embedding-based RTP**

**Option 1:**
Embed whole change at once

**Option 2:**
Embed chunks, use similarity to closest chunk

---

2024-03-12
Mattis, Böhme, Krebs, Rinard, Hirschfeld | Faster Feedback with AI? | Software Architecture Group
Evaluation

- Change-based mutation testing on open-source Python projects
- Compare performance of:
  - LLM
  - Embedding strategies
  - BM25 baseline

<table>
<thead>
<tr>
<th></th>
<th>Commits</th>
<th>Tests (Param.)</th>
<th>Faults</th>
<th>LOC Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flask</td>
<td>159</td>
<td>390 (442)</td>
<td>726</td>
<td>12.5</td>
</tr>
<tr>
<td>Requests</td>
<td>43</td>
<td>314 (557)</td>
<td>188</td>
<td>13.8</td>
</tr>
<tr>
<td>Jinja</td>
<td>68</td>
<td>655 (829)</td>
<td>420</td>
<td>15.2</td>
</tr>
</tbody>
</table>
Evaluation | Metrics

- **Performance**: average percentage of faults detected (APFD)
- Computes the area under the curve that plots the percentage of uncovered faults so far (y-axis) over the percentage of already executed tests (x-axis)
Results | Flask

Fault detection (flask)

- IIm (APFD = 0.890)
- embed (APFD = 0.923)
- embed (whole) (APFD = 0.922)
- bm25 (APFD = 0.931)
- rand (APFD = 0.709)
- default (APFD = 0.629)

Number of tests executed vs. Percentage of faults detected
### Results | Requests

![Fault detection (requests)](image)

<table>
<thead>
<tr>
<th>Method</th>
<th>APFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>llm</td>
<td>0.973</td>
</tr>
<tr>
<td>embed (APFD = 0.980)</td>
<td></td>
</tr>
<tr>
<td>embed (whole)</td>
<td>0.960</td>
</tr>
<tr>
<td>bm25 (APFD = 0.944)</td>
<td></td>
</tr>
<tr>
<td>rand (APFD = 0.861)</td>
<td></td>
</tr>
<tr>
<td>default</td>
<td>0.781</td>
</tr>
</tbody>
</table>
Results | Jinja

Fault detection (jinja)

- IIm (APFD = 0.874)
- embed (APFD = 0.934)
- embed (whole) (APFD = 0.896)
- bm25 (APFD = 0.863)
- rand (APFD = 0.817)
- default (APFD = 0.750)
Discussion

✓ (Chunked) embeddings work well

🚀 Competitiveness of simple BM25 model surprising

❌ LLMs get “distracted” and are slow
  - Probability depends on “coding style” and consistency
  - Computation of priority = up to 10x test execution
Future Work | **Fine-tuning**

警示 Study limited to pre-trained models

**LLM fine-tuning**
- 🌸 Low-rank adaptation (LoRA)
- ✅ Coding style
- ✅ Abstractions of the underlying project(s)
- ✅ Task-specific prompts

**Embedding fine-tuning**
- 🌸 Positive/negative examples from mutations
Future Work | **GAR**

**Generation-augmented Retrieval (GAR)**
- Generate “ideal” tests
- Embed together with real tests
- Choose tests with highest similarity to generated test

1. **Generation**

2. **Retrieval / Ranking**
Future Work | **Live Examples**

Example-based Live Programming (ELP):

- Concrete values directly at code-level
- Probes to sample intermediate results
- Live update after change

Prioritize examples rather than tests
1. A change-based mutation-testing framework to compare RTP strategies on three Python projects

2. RTP benefits from embeddings, but the simpler BM25 performs well

3. LLMs of limited use; fine-tuning or GAR promising next steps
Backup Slides
def test_scriptinfo(test_apps, monkeypatch):
    obj = ScriptInfo(app_import_path="cliapp.app:testapp")
    app = obj.load_app()
    assert app.name == "testapp"
    assert obj.load_app() is app

    # import app with module’s absolute path
    cli_app_path = str(test_path / "cliapp" / "app.py")
    obj = ScriptInfo(app_import_path=cli_app_path)
    app = obj.load_app()
    assert app.name == "testapp"
    assert obj.load_app() is app

    def create_app():
        return Flask("createapp")

    obj = ScriptInfo(create_app=create_app)
    app = obj.load_app()
    assert app.name == "createapp"
    assert obj.load_app() is app

    pytest.raises(NoAppException, obj.load_app)

    # import app from wsgi.py in current directory
    monkeypatch.chdir(test_path / "helloworld")
    obj = ScriptInfo()
    app = obj.load_app()
    assert app.name == "hello"

    # import app from app.py in current directory
    monkeypatch.chdir(test_path / "cliapp")
    obj = ScriptInfo()
    app = obj.load_app()
    assert app.name == "testapp"
## Test LOC

<table>
<thead>
<tr>
<th></th>
<th>Commits</th>
<th>Test methods (Tests run)</th>
<th>Faults</th>
<th>LOC Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flask</td>
<td>159</td>
<td>390 (442)</td>
<td>726</td>
<td>12.5</td>
</tr>
<tr>
<td>Requests</td>
<td>43</td>
<td>314 (557)</td>
<td>188</td>
<td>13.8</td>
</tr>
<tr>
<td>Jinja</td>
<td>68</td>
<td>655 (829)</td>
<td>420</td>
<td>15.2</td>
</tr>
</tbody>
</table>

### Test sizes:

![Flask Test Size](image)

![Requests Test Size](image)

![Jinja Test Size](image)
Change-based Mutation Testing

1. `def run_change_based_mutation_testing():`
2. `repo = git.Repo('flask')`
3. `for commit in repo.commits():`

* pseudocode only
Change-based Mutation Testing

1. def run_change_based_mutation_testing():
2.     repo = git.Repo('flask')
3.     for commit in repo.commits():
4.         change_diff = diff(commit.parent, commit)
Change-based Mutation Testing

1. ```python
def run_change_based_mutation_testing():
    repo = git.Repo('flask')
    for commit in repo.commits():
        change_diff = diff(commit.parent, commit)

    # Start experiment
    commit.checkout()
    commit.repo.install_dependencies()
    control_test_results = run_tests(commit)
```
Change-based Mutation Testing

1. `def run_change_based_mutation_testing():`
2. `    repo = git.Repo('flask')`
3. `    for commit in repo.commits():`
4. `        change_diff = diff(commit.parent, commit)`
5. `        # Start experiment`
6. `        commit.checkout()`
7. `        commit.repo.install_dependencies()`
8. `        control_test_results = run_tests(commit)`
9. `        # Get mutations`
10. `        mutation_sites = get_mutation_sites(change_diff)`

<table>
<thead>
<tr>
<th>Mutation</th>
<th>Change</th>
<th>Defect injected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>$1 + 3$</td>
<td>$1 - 3$</td>
</tr>
<tr>
<td>Number</td>
<td><code>year = 2024</code></td>
<td><code>year = 42</code></td>
</tr>
<tr>
<td>String</td>
<td>“Lund is awesome”</td>
<td>prompt = “”</td>
</tr>
<tr>
<td>Condition</td>
<td>if conferene.started:</td>
<td>if True:</td>
</tr>
</tbody>
</table>

* pseudocode only
Change-based Mutation Testing

1. `def run_change_based_mutation_testing():`
2. `    repo = git.Repo('flask')`
3. `    for commit in repo.commits():`
4. `        change_diff = diff(commit.parent, commit)`
5. `        # Start experiment`
6. `        commit.checkout()`
7. `        commit.repo.install_dependencies()`
8. `        control_test_results = run_tests(commit)`
9. `        # Get mutations`
10. `        mutation_sites = get_mutation_sites(change_diff)`
11. `        # Apply mutation sites and run tests on mdefected commits`
12. `        mutation_results = []`
13. `        for mutation_site in mutation_sites:`
14. `            commit.checkout()`
15. `            defected_commit = apply_mutation(commit, mutation_site)`
16. `            mutation_results += run_tests(defected_commit)`
17. `        * pseudocode only`
Results - Flask

Fault detection (flask)

prioritization strategy

apfd

flask

Number of tests executed

Percentage of faults detected

IIm (APFD = 0.890)
embed (APFD = 0.923)
bm25 (APFD = 0.931)
rand (APFD = 0.709)
default (APFD = 0.629)
Results - Jinja

Fault detection (jinja)

- Ilm (APFD = 0.874)
- embed (APFD = 0.934)
- bm25 (APFD = 0.863)
- rand (APFD = 0.817)
- default (APFD = 0.750)