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

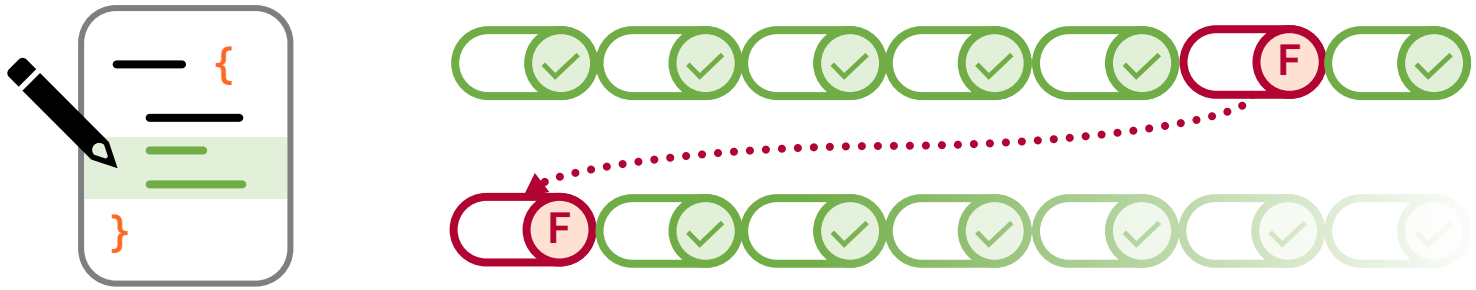
PAI/24 | Lund, Sweden | March 12

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

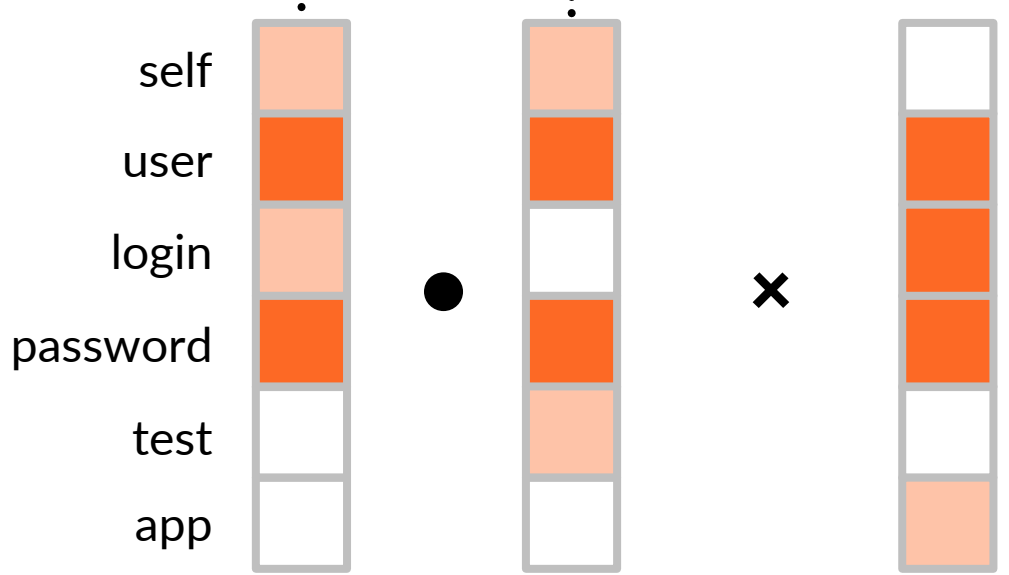
1. Semantic similarity: TF-IDF
2. Semantic similarity: Embeddings
3. Large Language Models

Background | **Semantic Similarity**

- Traditional approach: shared vocabulary via TF-IDF

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

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



Term frequency (TF)

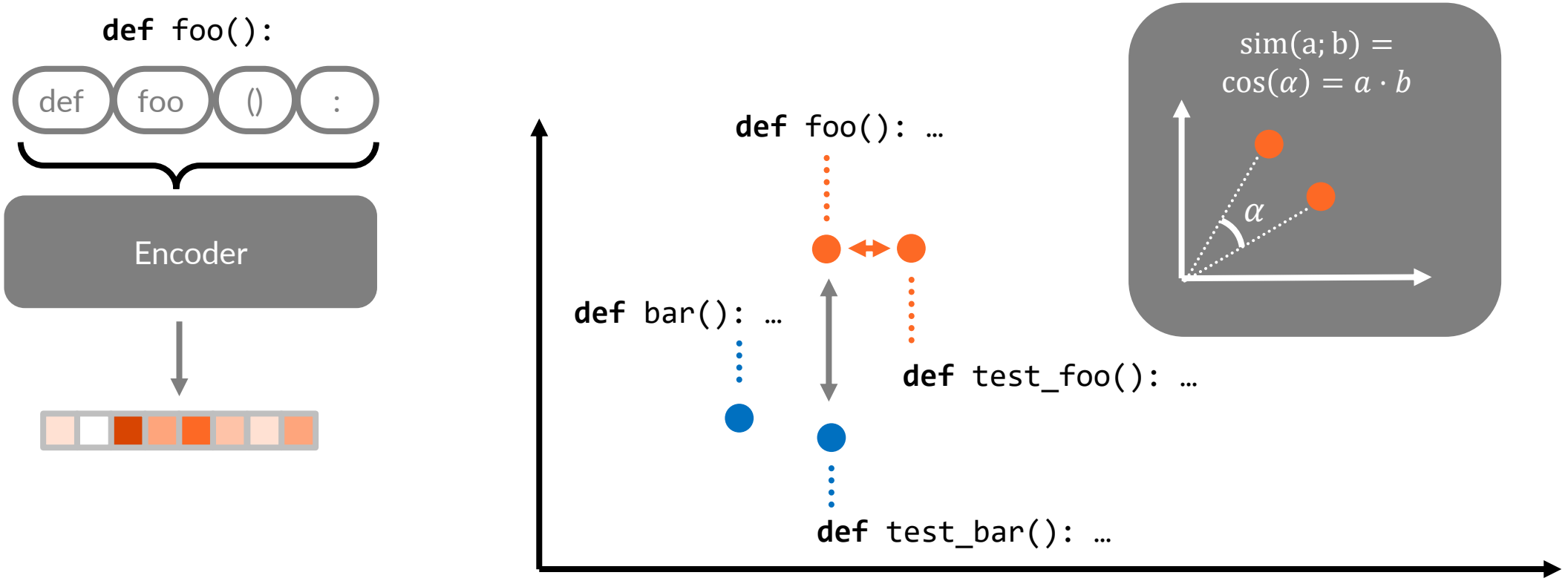
Term importance

Inverse Document Frequency (IDF)

Here: **Okapi BM25** (from Apache Lucene)

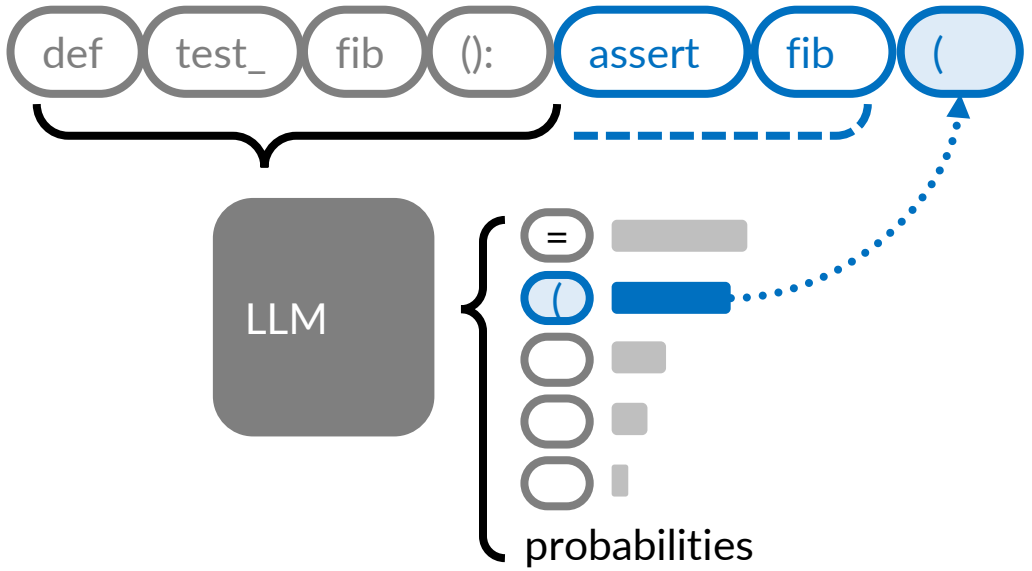
Background | **Embeddings**

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



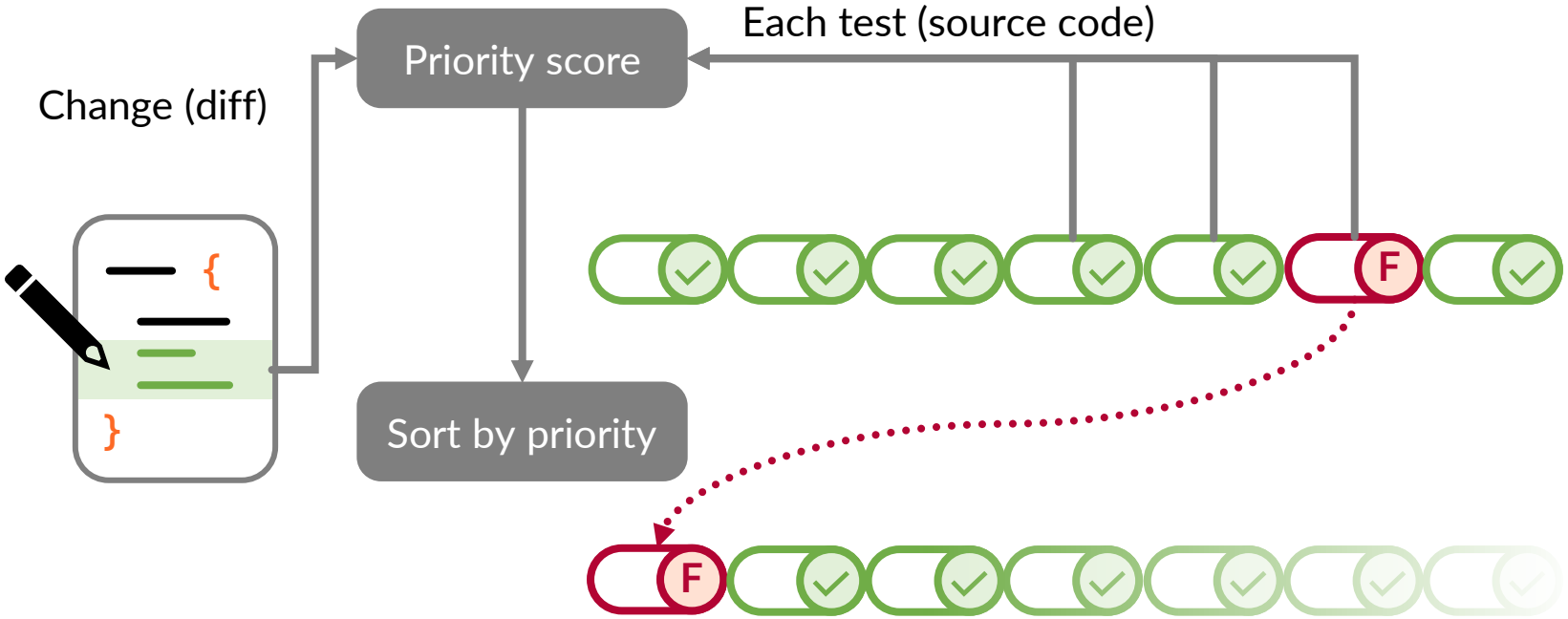
Background | **Language Models**

- 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



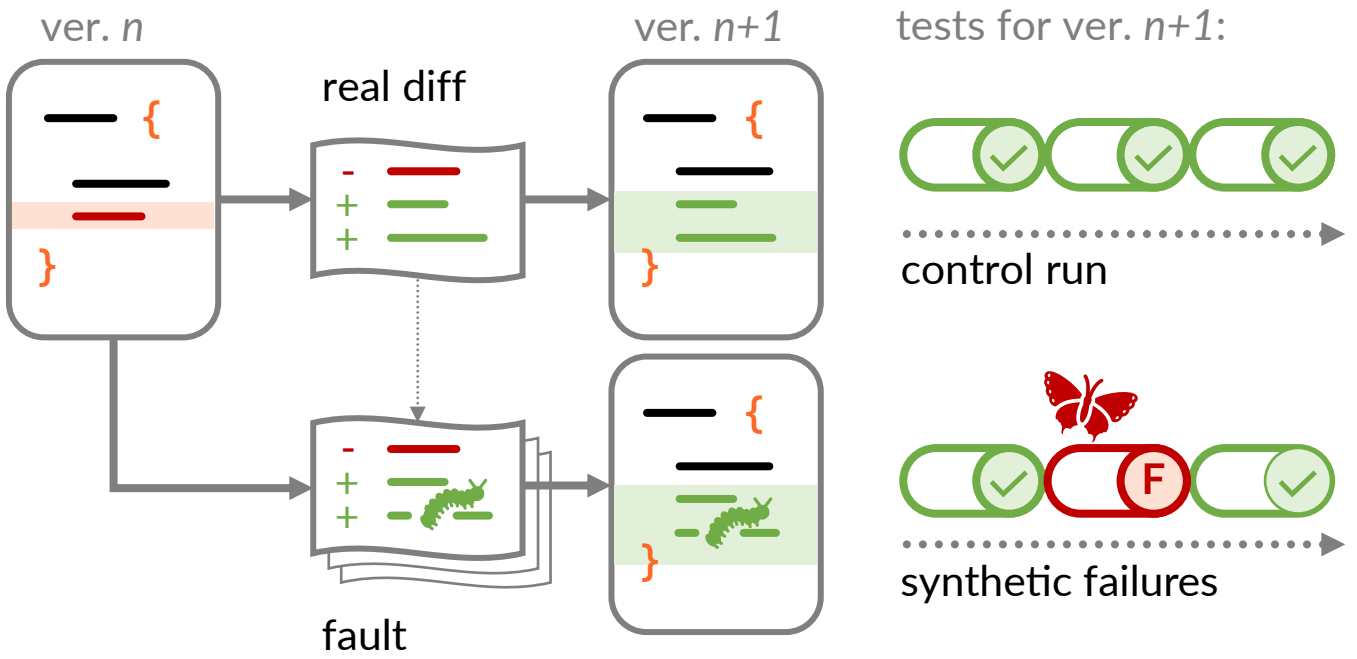
Approach | **AI-based Test Prioritization**

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



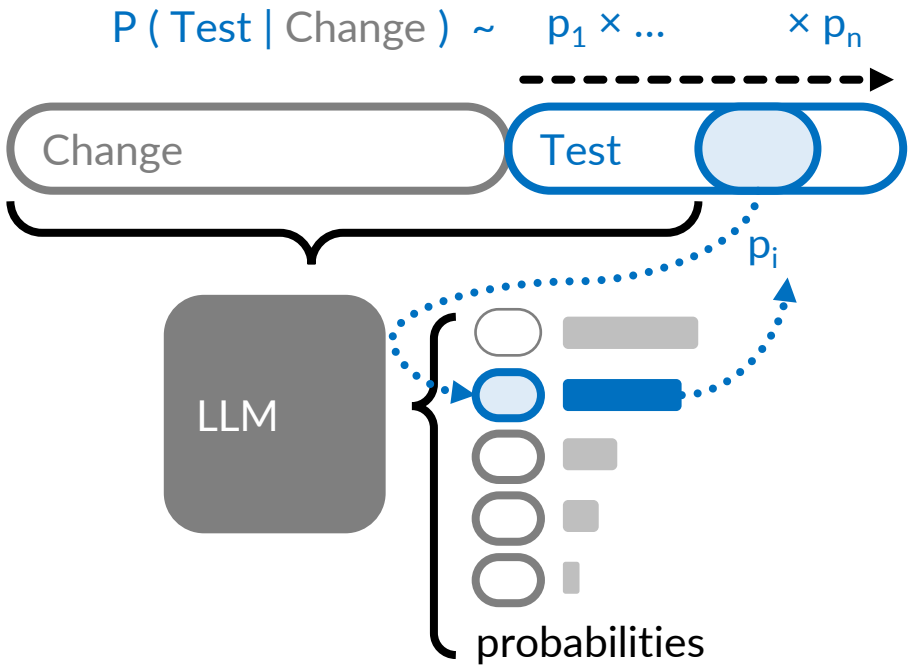
Approach | **Change-based Mutation Testing**

Mutation	Change	Defect injected
Binary	`1 + n`	`1 - n`
	`a * b`	`a / b`
Number	`year = 2024`	`year = 42`
String	`prompt = “Lund is awesome”`	`prompt = “”`
Condition	`if conference.started:`	`if True:`
		`if False:`

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

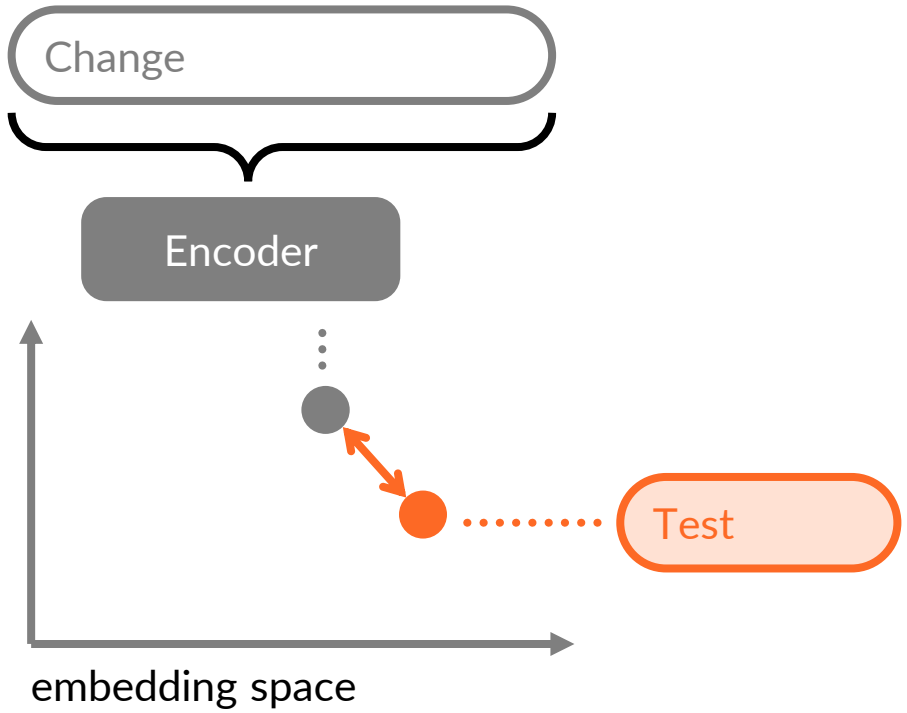
```
class AClass:

    def __init__(self):
        …

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

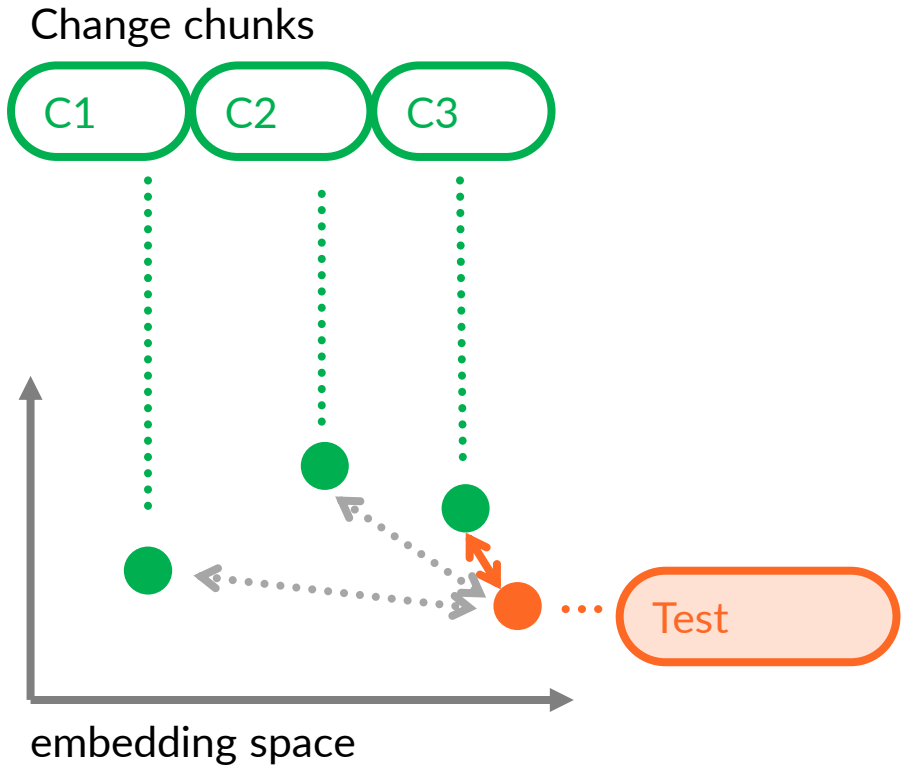
Change

```
# /src/the_file.py
class AClass:

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

# Test validating this change:
```

LLM Prompt

Approach | **Embedding-based RTP**





Option 1:
Embed whole change at once

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

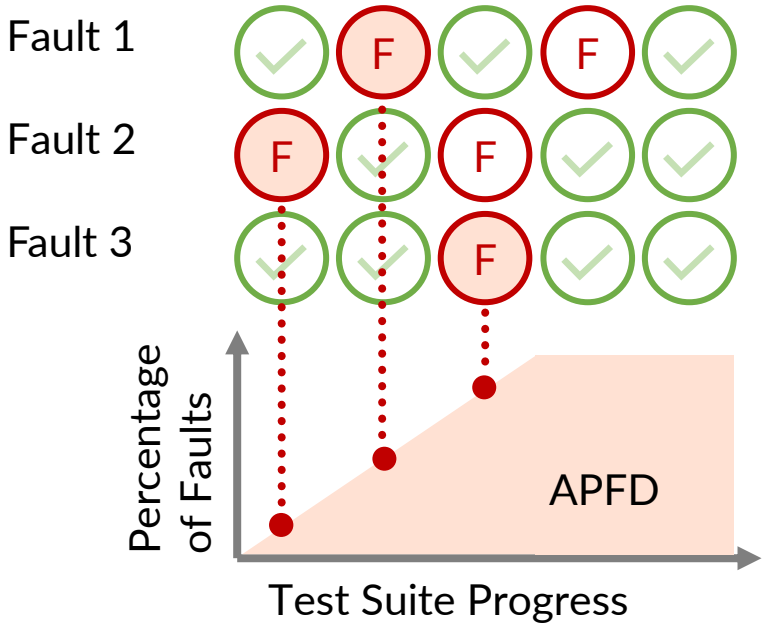
Evaluation

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

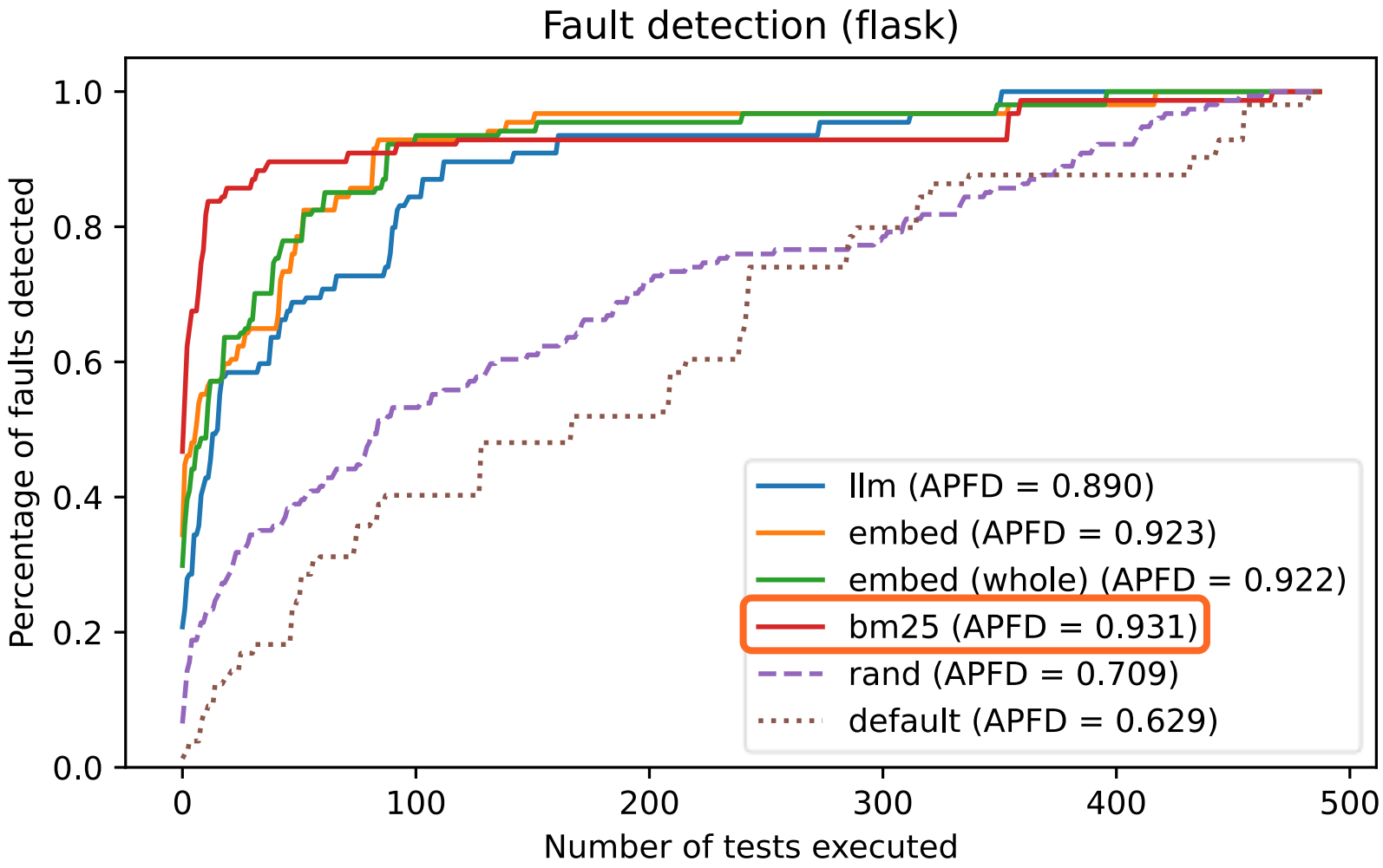
	Commits	Tests (Param.)	Faults	LOC Changed
Flask	159	390 (442)	726	12.5
Requests	43	314 (557)	188	13.8
Jinja	68	655 (829)	420	15.2

Evaluation | **Metrics**

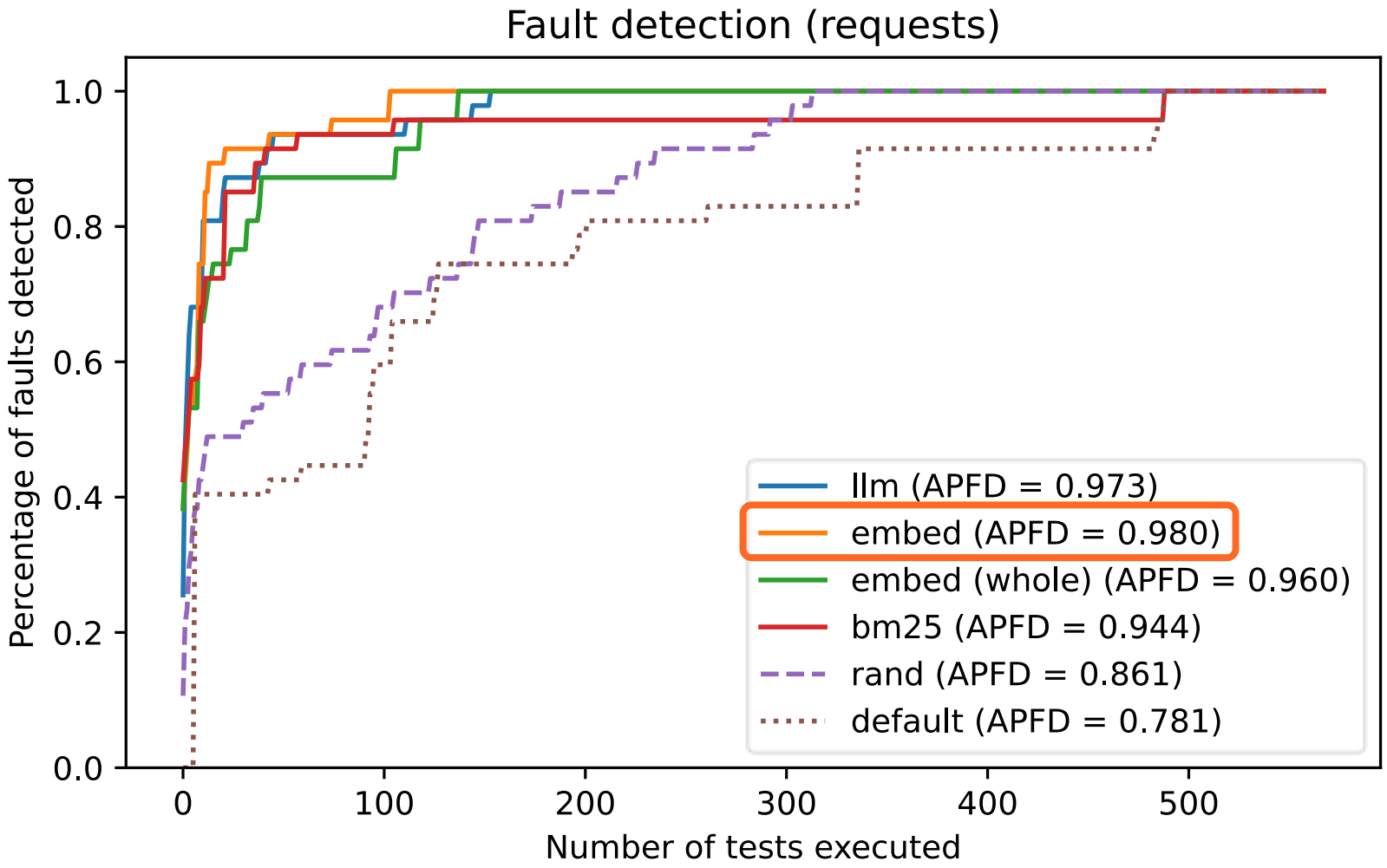
- **Performance:** average percentage of faults detected (APFD)
- Computes the area under the curve that plots the percentage of **uncovered faults so far** (y-axis) over the percentage of **already executed tests** (x-axis)



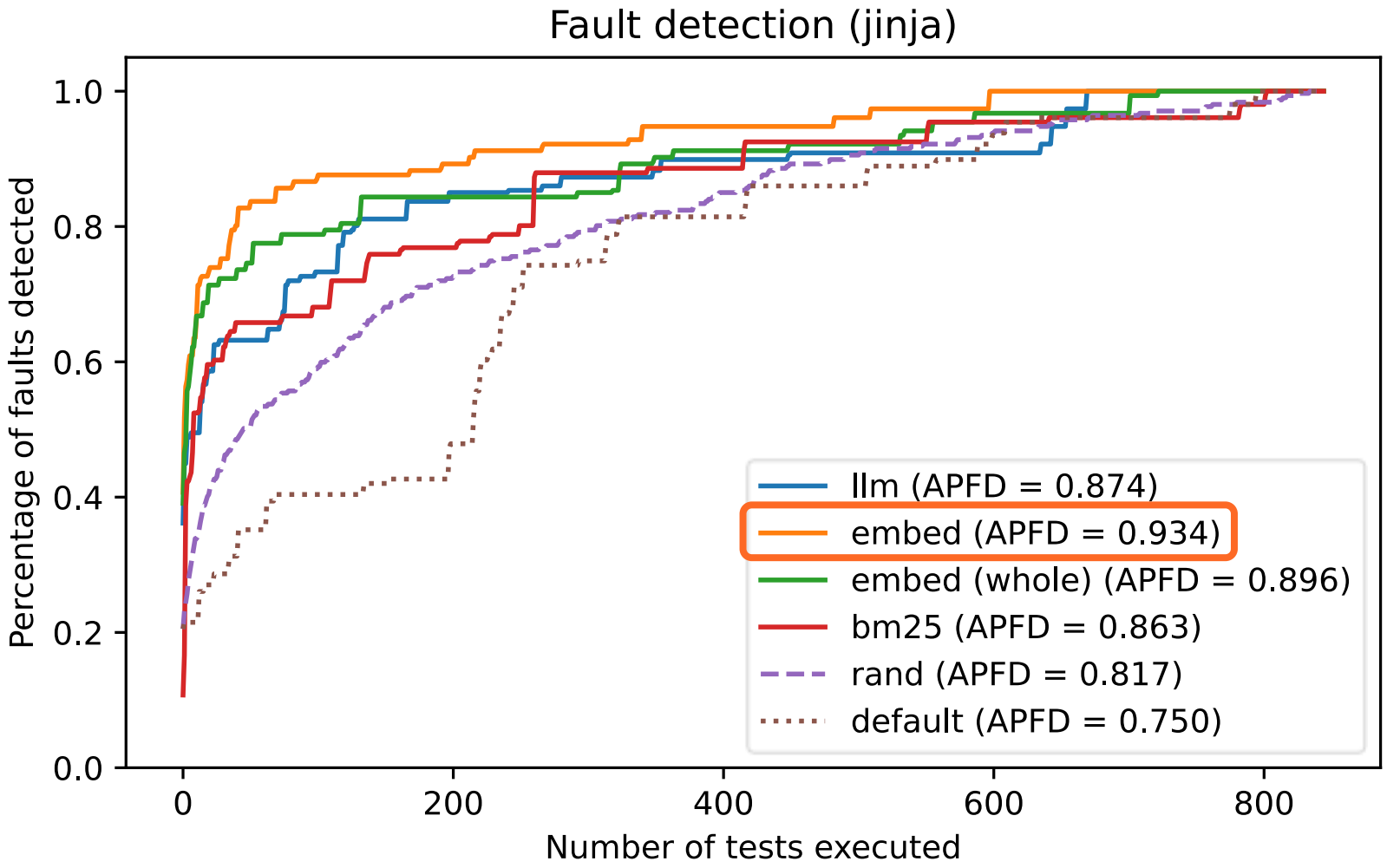
Results | **Flask**



Results | **Requests**



Results | **Jinja**



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**

⚡ Study limited to pre-trained models

LLM fine-tuning

- ⚙️ Low-rank adaptation (LoRA)
- ✅ Coding style
- ✅ Abstractions of the underlying project(s)
- ✅ Task-specific prompts

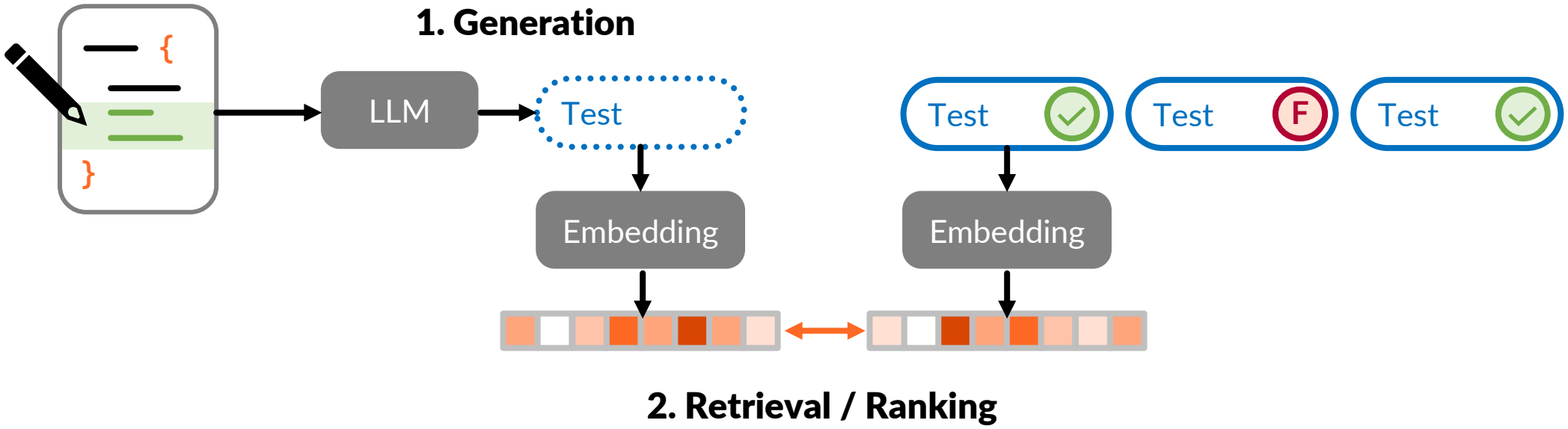
Embedding fine-tuning

- ⚙️ Positive/negative examples from mutations

Future Work | **GAR**

Generation-augmented Retrieval (GAR)

- Generate “ideal” tests
- Embed together with real tests
- Chose tests with highest similarity to generated test

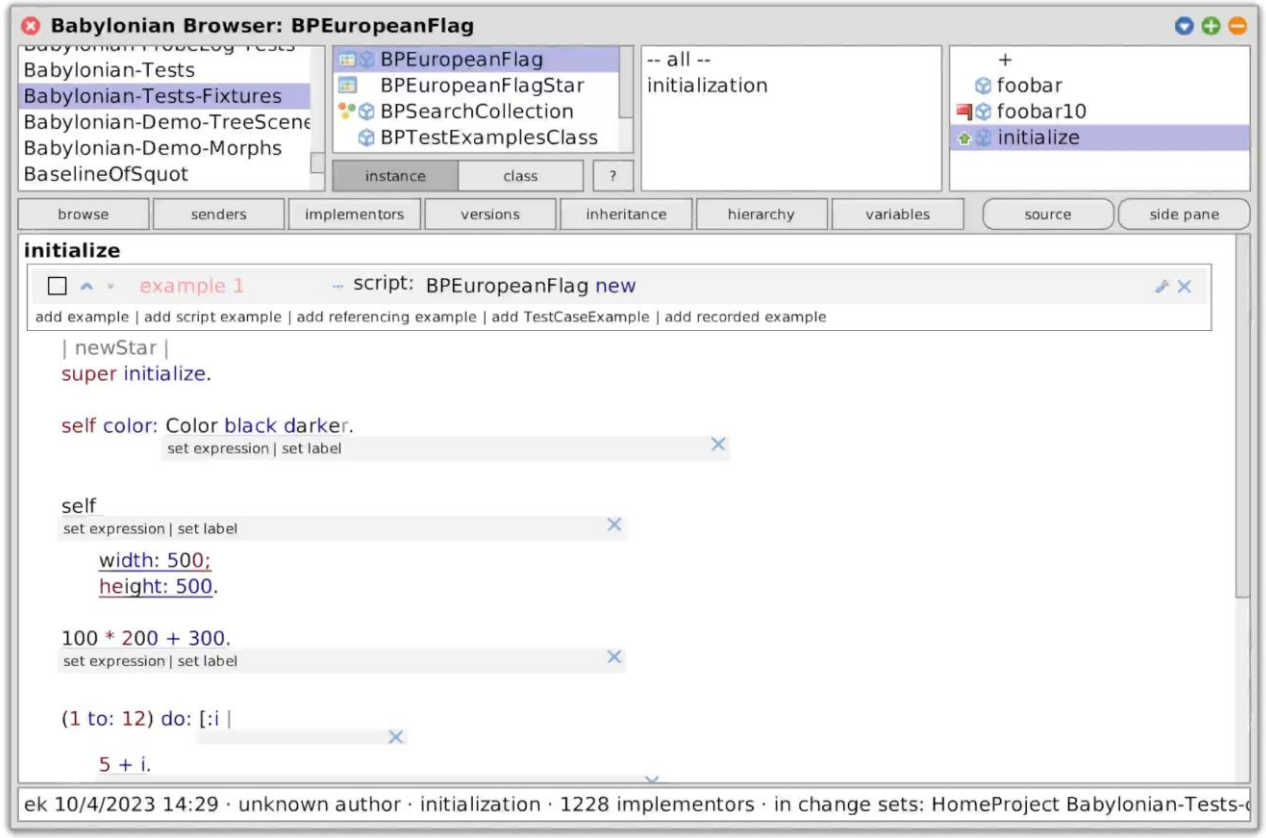


Future Work | **Live Examples**

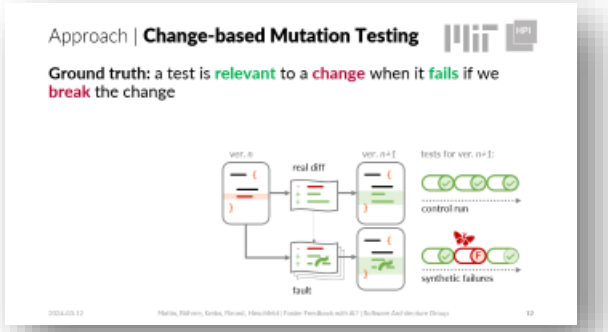
Example-based Live Programming (ELP):

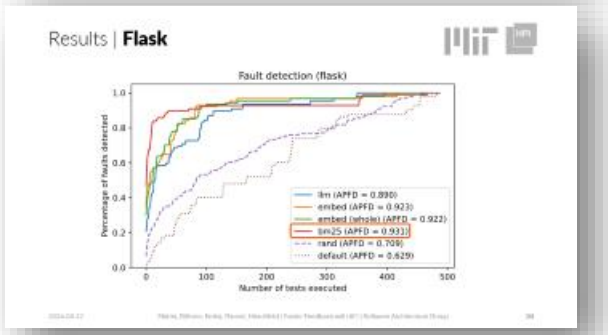
- Concrete values directly at code-level
- Probes to sample intermediate results
- Live update after change

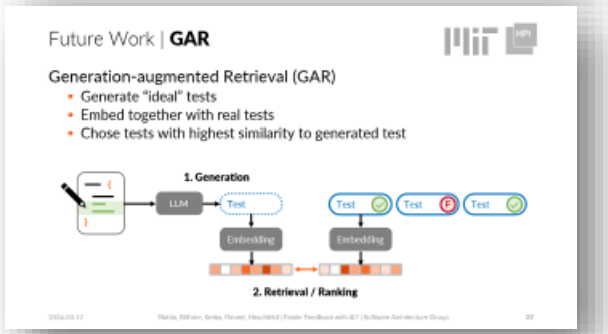
Prioritize **examples** rather than tests



Summary







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

LLM “Distraction”

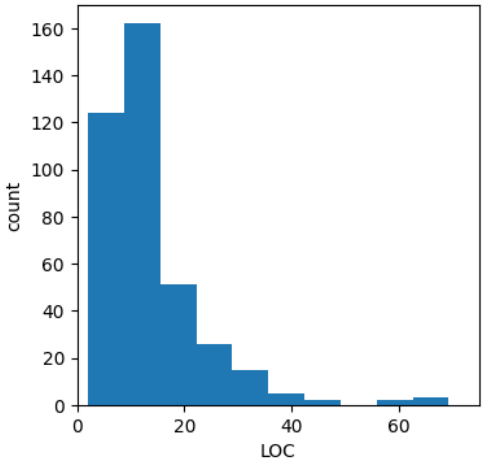
```python
def test_scriptinfo(test_apps, monkeypatch):
    obj = ScriptInfo(app_import_path="cliapp.app:testapp")
    app = obj.load_app()
    assert app.name == "testapp"
    assert obj.load_app() is app

    # 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
    obj = ScriptInfo(app_import_path=f"{cli_app_path}:testapp")
    app = obj.load_app()
    assert app.name == "testapp"
    assert obj.load_app() is app
```

```python
    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

    obj = ScriptInfo()
    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"
```

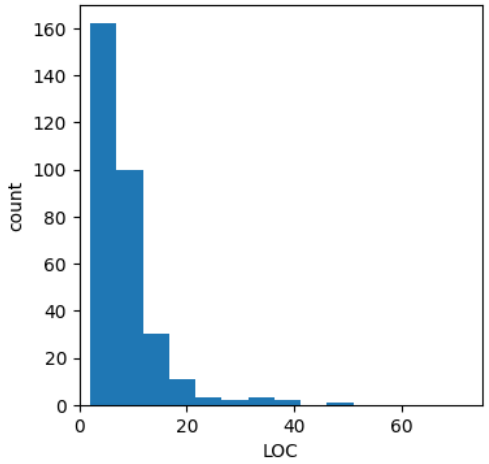
Test LOC

	Commits	Test methods (Tests run)	Faults	LOC Changed
Flask	159	390 (442)	726	12.5
Requests	43	314 (557)	188	13.8
Jinja	68	655 (829)	420	15.2

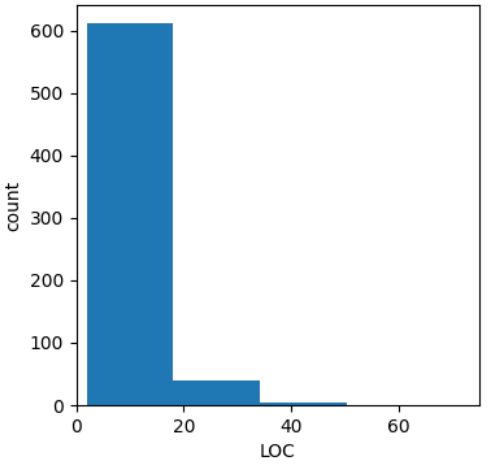
Test sizes:



Flask



Requests



Jinja

Change-based Mutation Testing


```
1. def run_change_based_mutation_testing():
2.     repo = git.Repo('flask')
3.     for commit in repo.commits():
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
```

* 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.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
```

* 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.
6.       # Start experiment
7.       commit.checkout()
8.       commit.repo.install_dependencies()
9.       control_test_results = run_tests(commit)
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
```

* 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.
6.       # Start experiment
7.       commit.checkout()
8.       commit.repo.install_dependencies()
9.       control_test_results = run_tests(commit)
10.
11.      # Get mutations
12.      mutation_sites = get_mutation_sites(change_diff)
13.
14.
15.
16.
17.
18.
19.
20.
```

Mutation	Change	Defect injected
Binary	1 + 3	1 - 3
Number	year = 2024	year = 42
String	“Lund is awesome”	prompt = “”
Condition	if conferene.started:	if True:

Types of Mutations

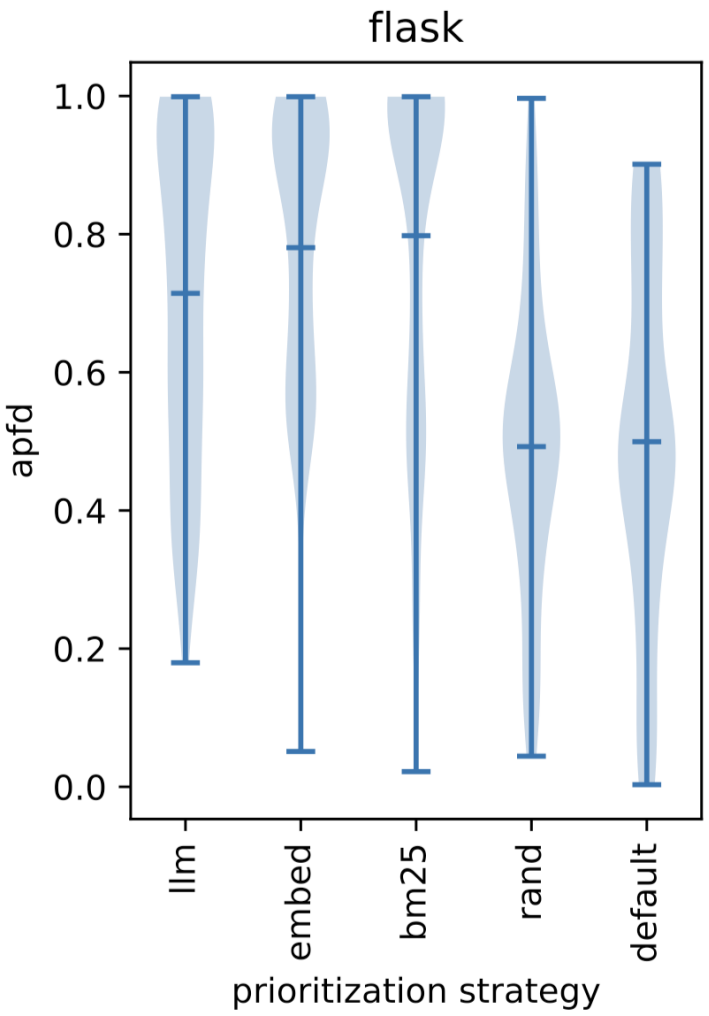
* pseudocode only

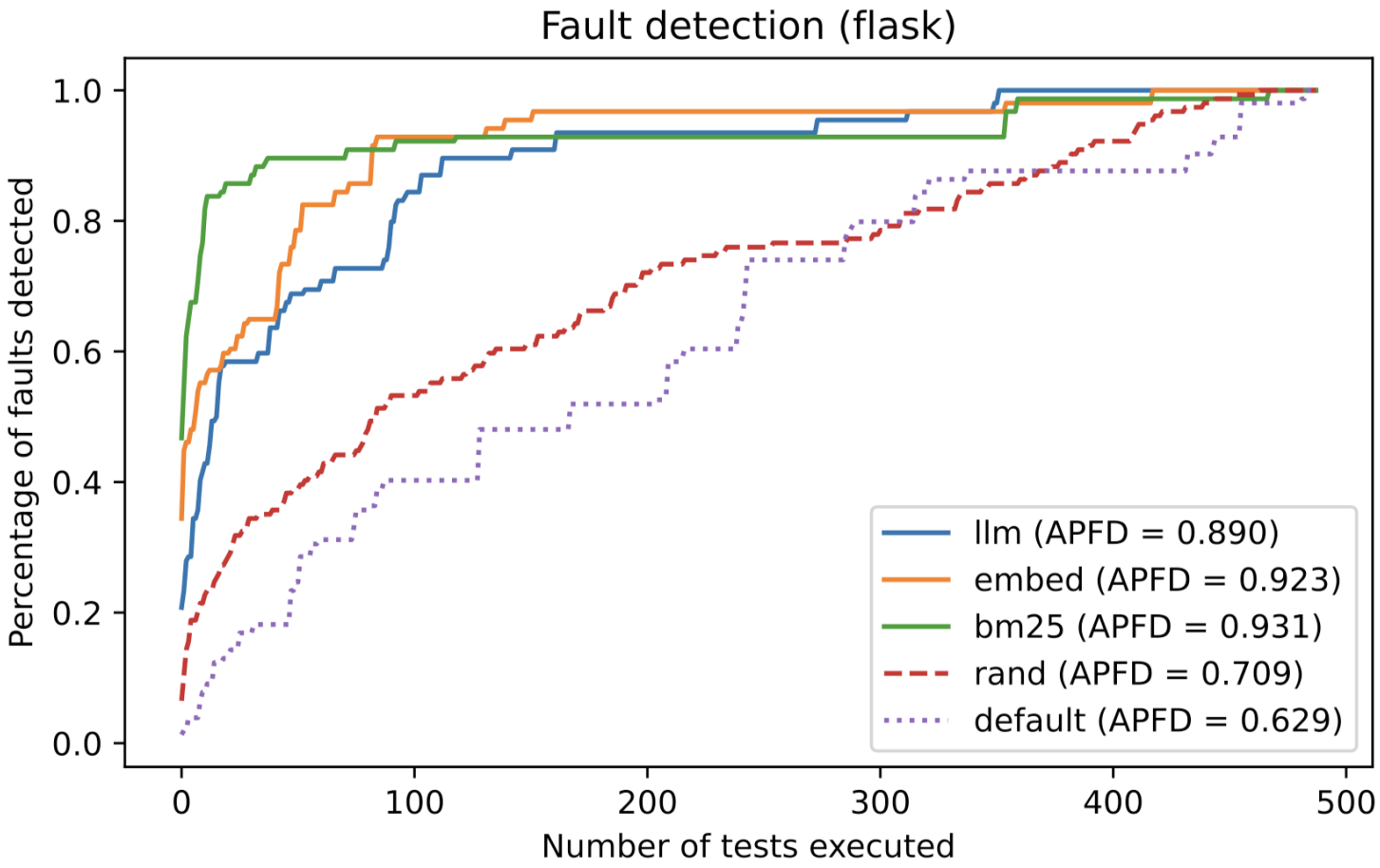
Change-based Mutation Testing


```
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.
6.       # Start experiment
7.       commit.checkout()
8.       commit.repo.install_dependencies()
9.       control_test_results = run_tests(commit)
10.
11.      # Get mutations
12.      mutation_sites = get_mutation_sites(change_diff)
13.
14.      # Apply mutation sites and run tests on mdefected commits
15.      mutation_results = []
16.      for mutation_site in mutation_sites:
17.        commit.checkout()
18.        defected_commit = apply_mutation(commit, mutation_site)
19.        mutation_results += run_tests(defected_commit)
20.
```

* pseudocode only

Results - Flask





Results - Jinja