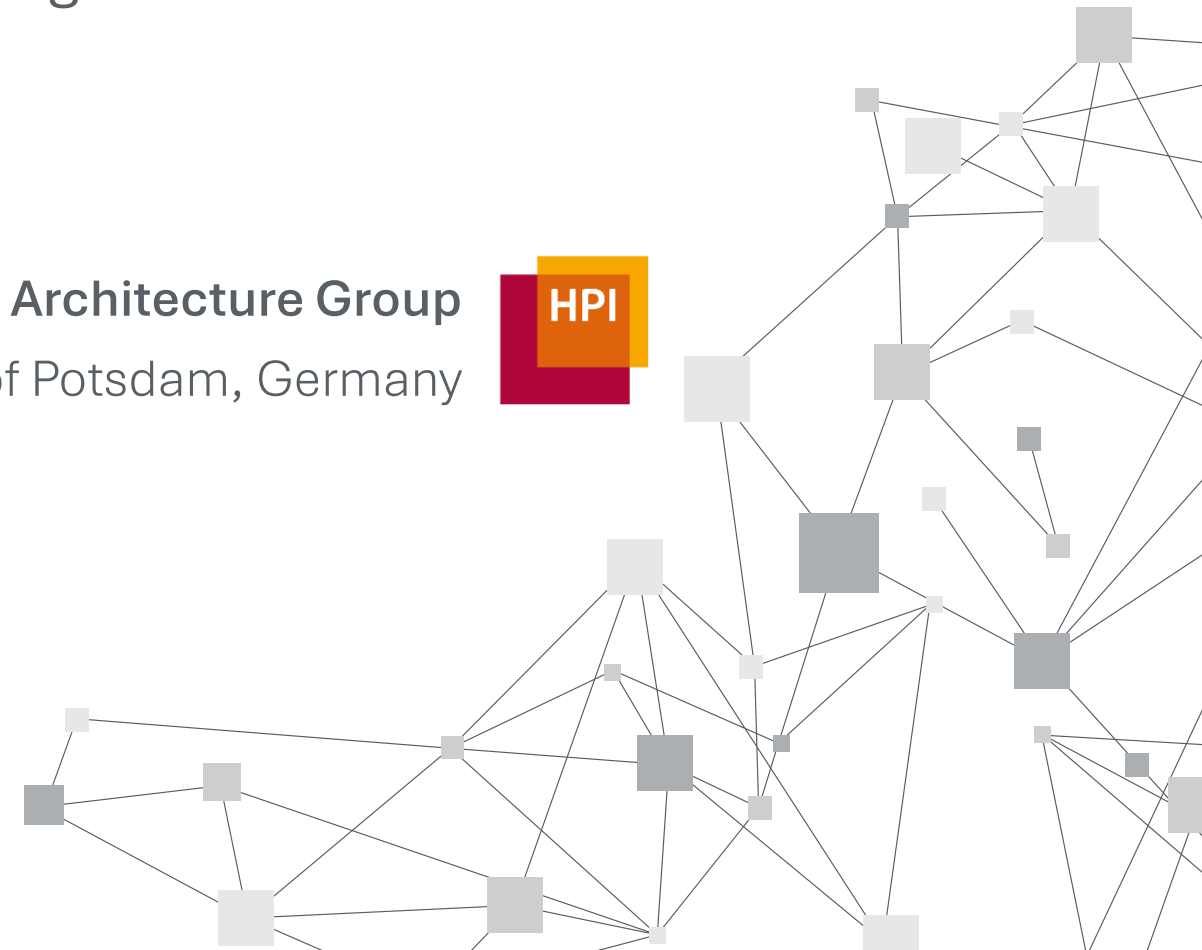
What can go wrong if I change this line?

Test Failure Prediction Using Natural Language and AI

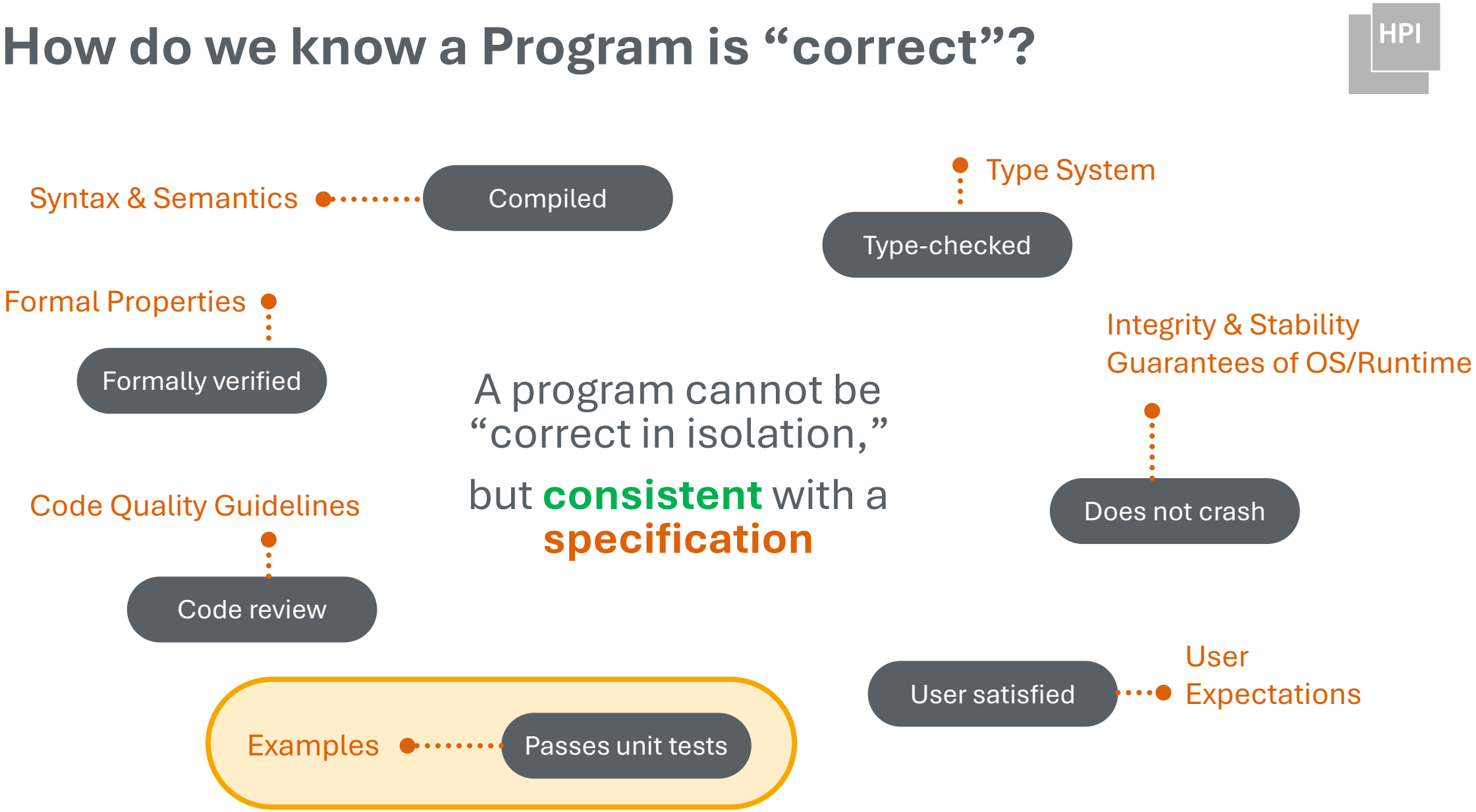
Toni Mattis
Robert Hirschfeld

Software Architecture Group
HPI, University of Potsdam, Germany



Potsdam | 14. Nov. 2024

How do we know a Program is “correct”?



Automated Testing

Unit under test

```
def with_tax(country, amount):
    return amount * (    VAT[country])
```

Unit test

```
def test_tax_germany():
    result = with_tax(countries.DE, 100)
    assert result == 119
```

Expected outputs and behavior

Example inputs and usage

“Specification by example”

“Executable documentation”

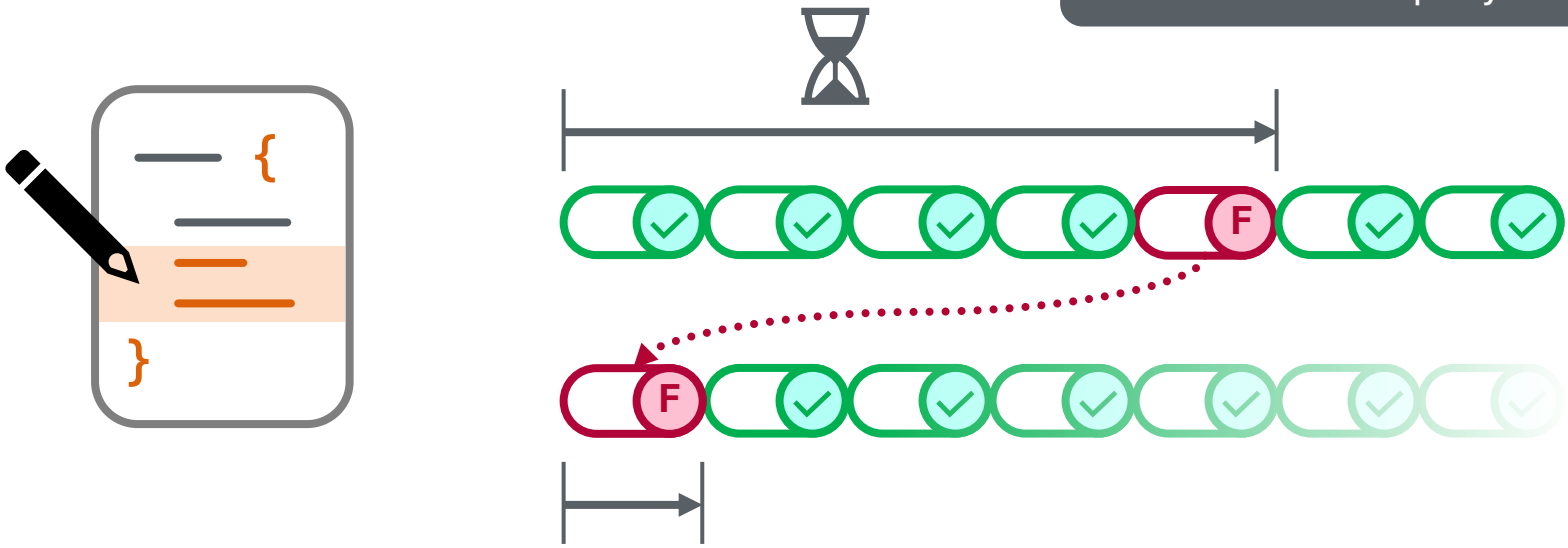
Tests and Feedback

✅ Test feedback helps **catch defects** early

⚡ Running many tests **delays** feedback

Goal: Run relevant tests first.

Flask: 442 Tests, a few minutes
Berlin-based company: 279 Tests, **18h(!)** [1]



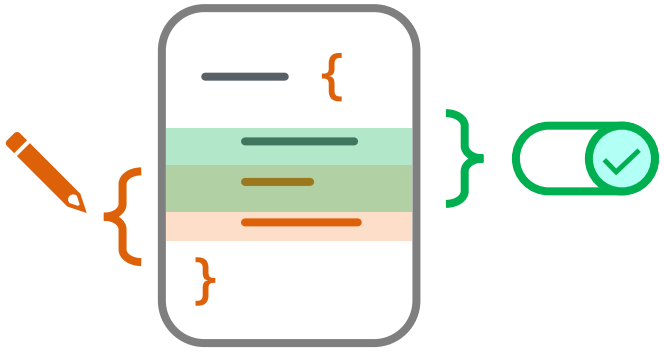
[1] Elsner et al. 2021: https://dl.acm.org/doi/abs/10.1145/3460319.3464834

The Test Prioritization Problem

How do we know which tests are **relevant*** to a change?

*) fail if the change introduces a defect (regression)

Main Approaches



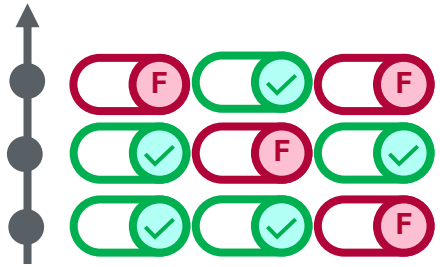
Coverage

Prioritize tests that cover code overlapping with the change by program analysis (static) or running tests (dynamic)

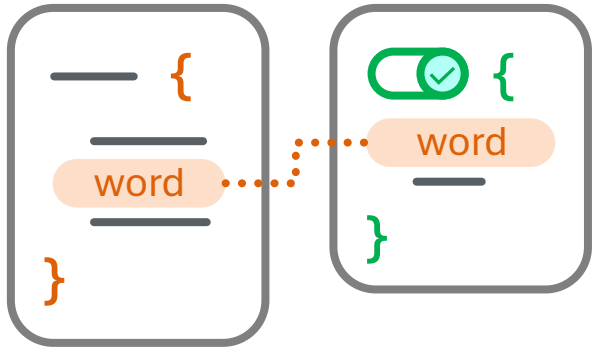
- Dynamic languages
- Becomes outdated

History

Prioritize tests that failed previously (for similar changes)



- Requires plenty of data



Natural Language

Prioritize tests with vocabulary / concepts overlapping with the change

Natural Language

Programming: “explaining to other programmers (+ your future self) how to make the computer solve a problem”

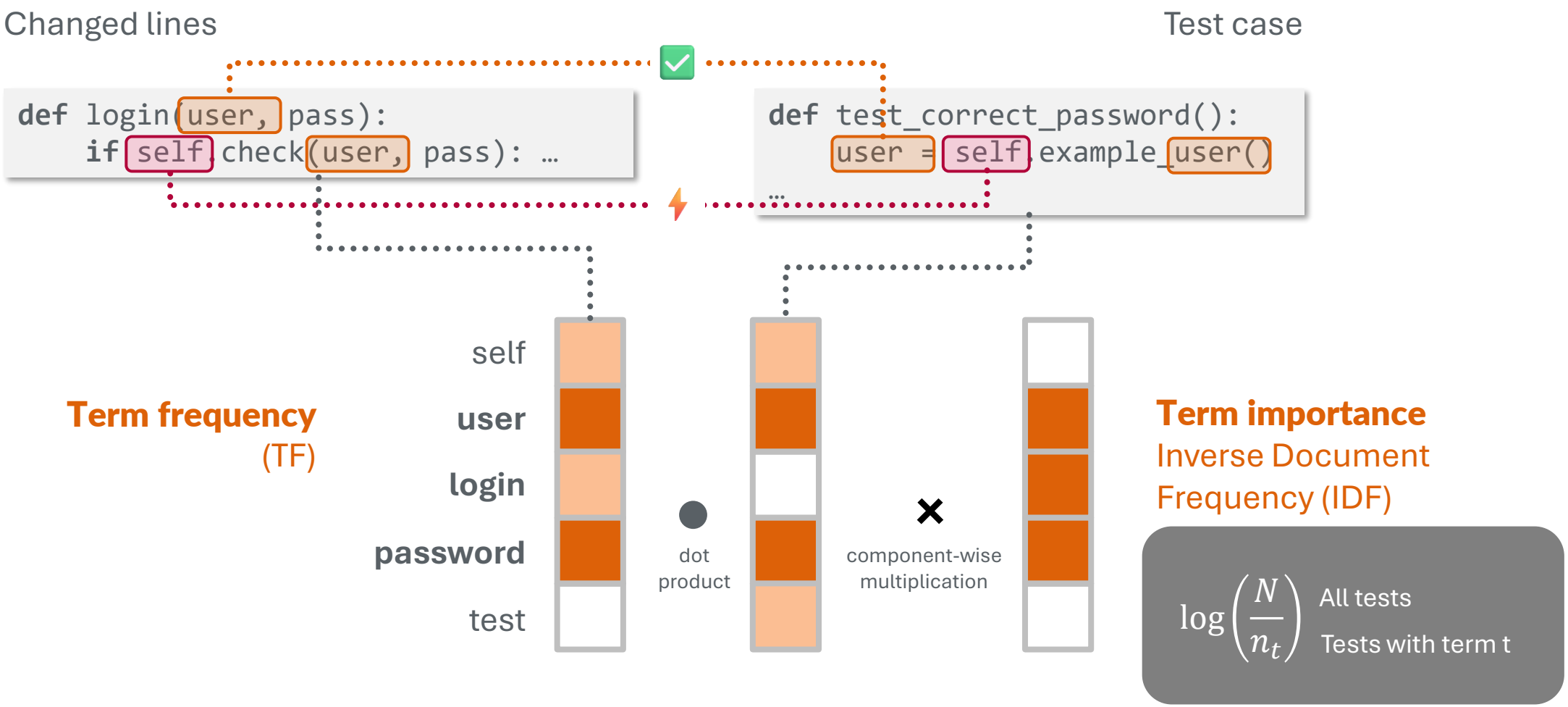
How to compute?
(similarity of natural language)

What to include?
(context surrounding change)

“Prioritize tests that share vocabulary with the changed code”

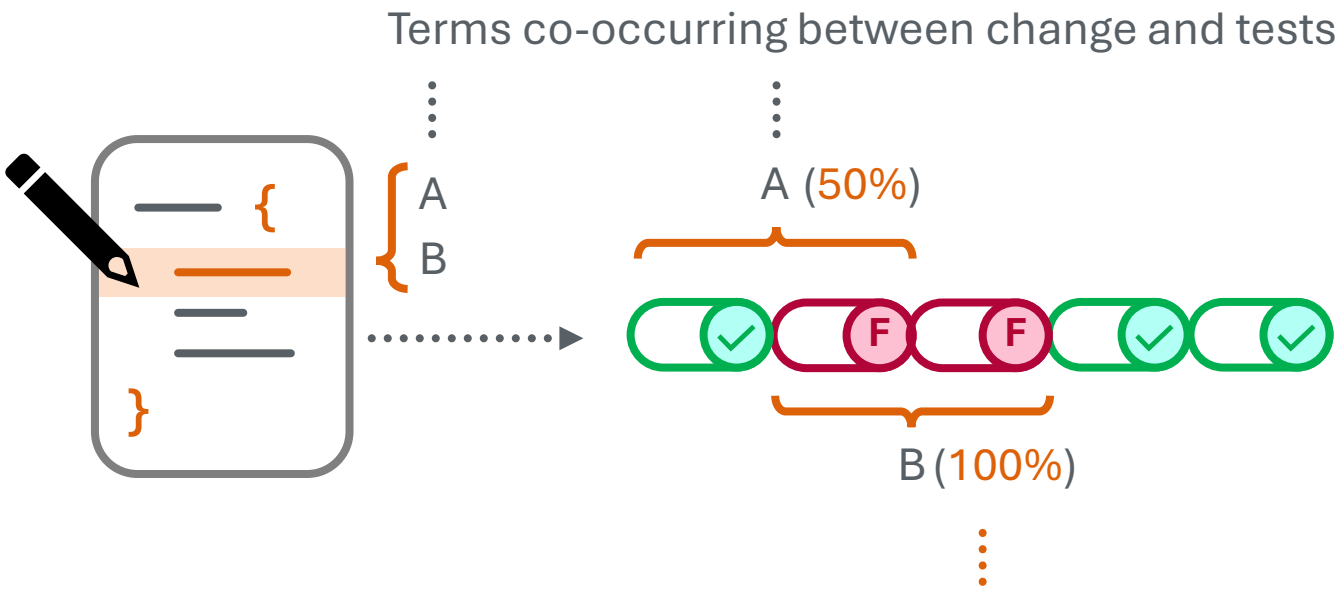
Which exactly?
(relevance of words)

Term Importance | **TF-IDF**



Term Importance | **Predictive Value**

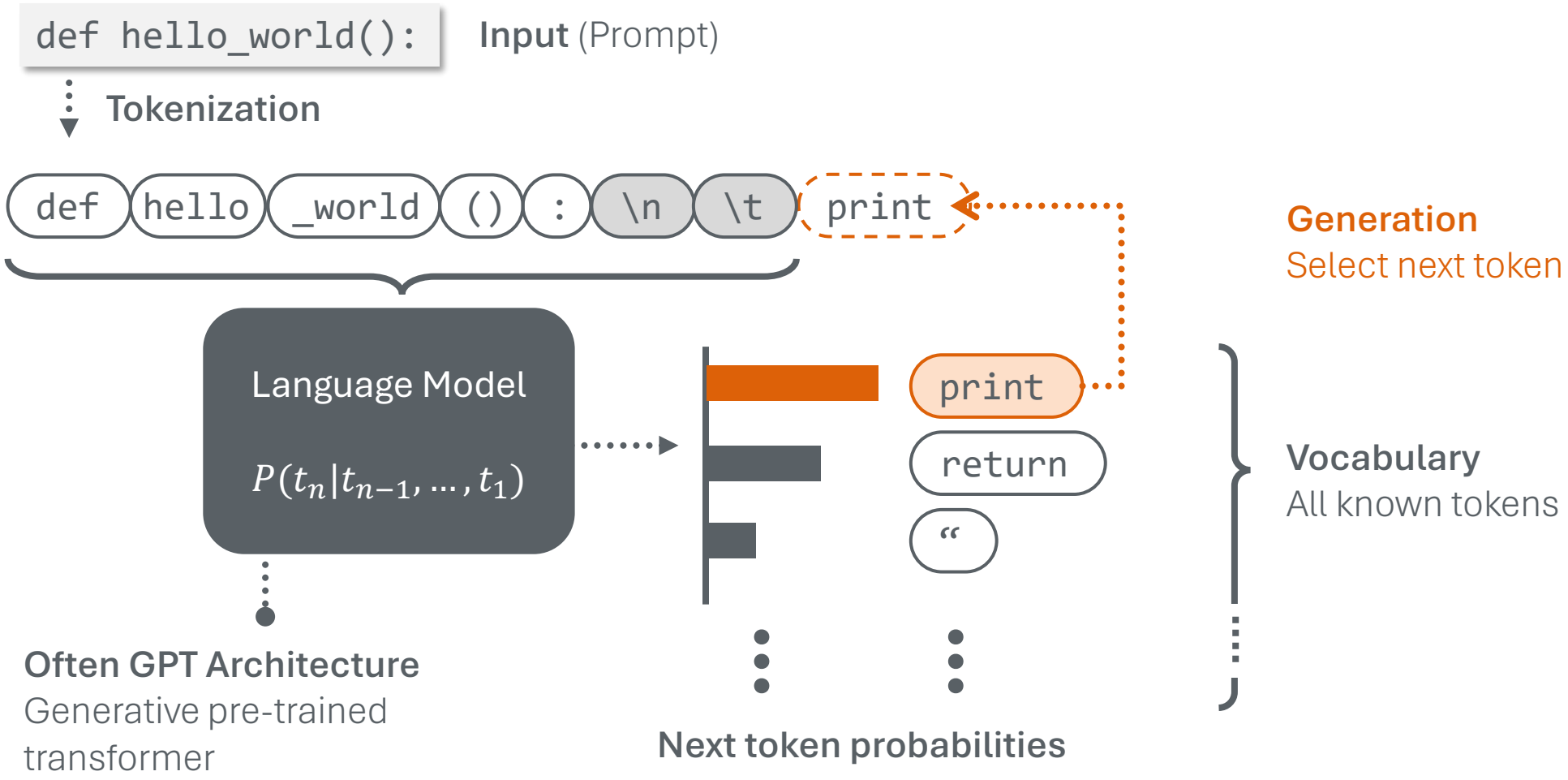
Weight terms according to how well they predicted previous test failures (precision)



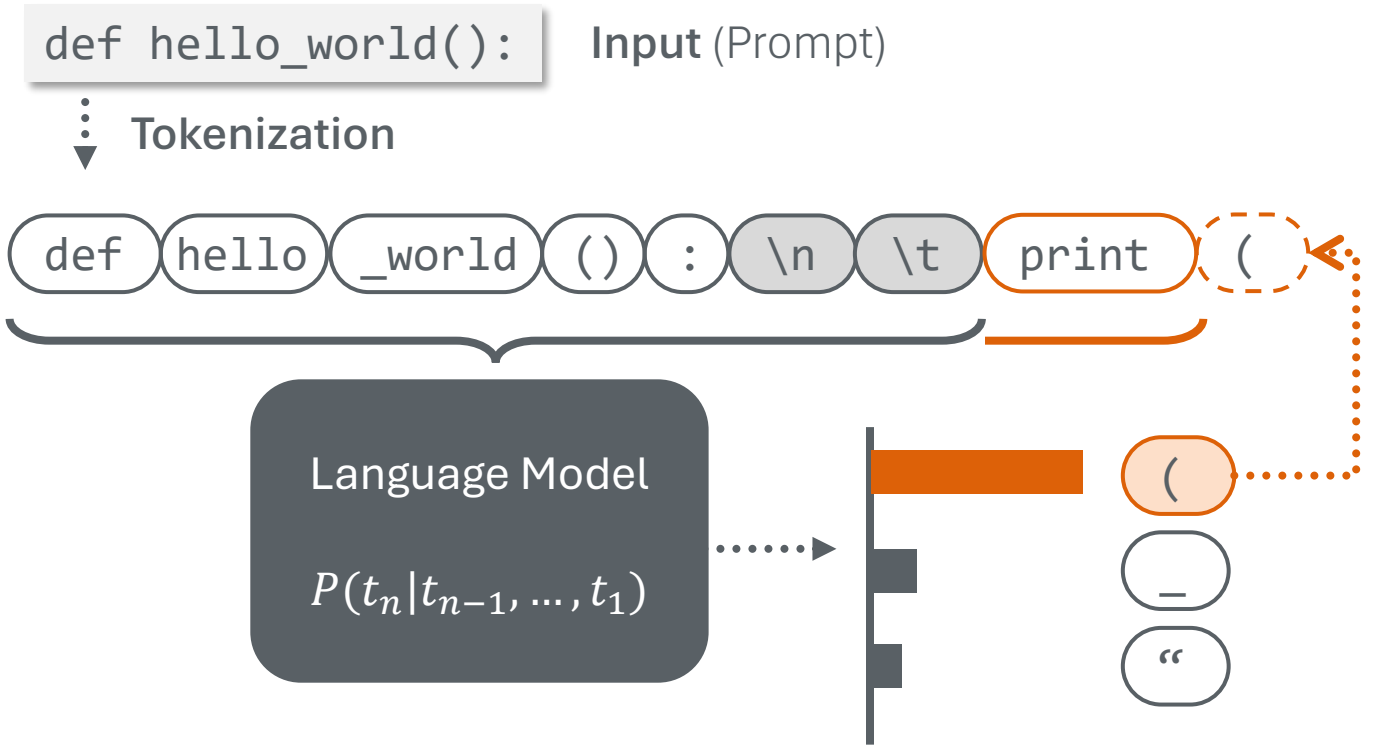
Without history: IDF
With history: Precision

Precision = #fails / #tests in which a term co-occurs with a change (replaces IDF in TF-IDF)

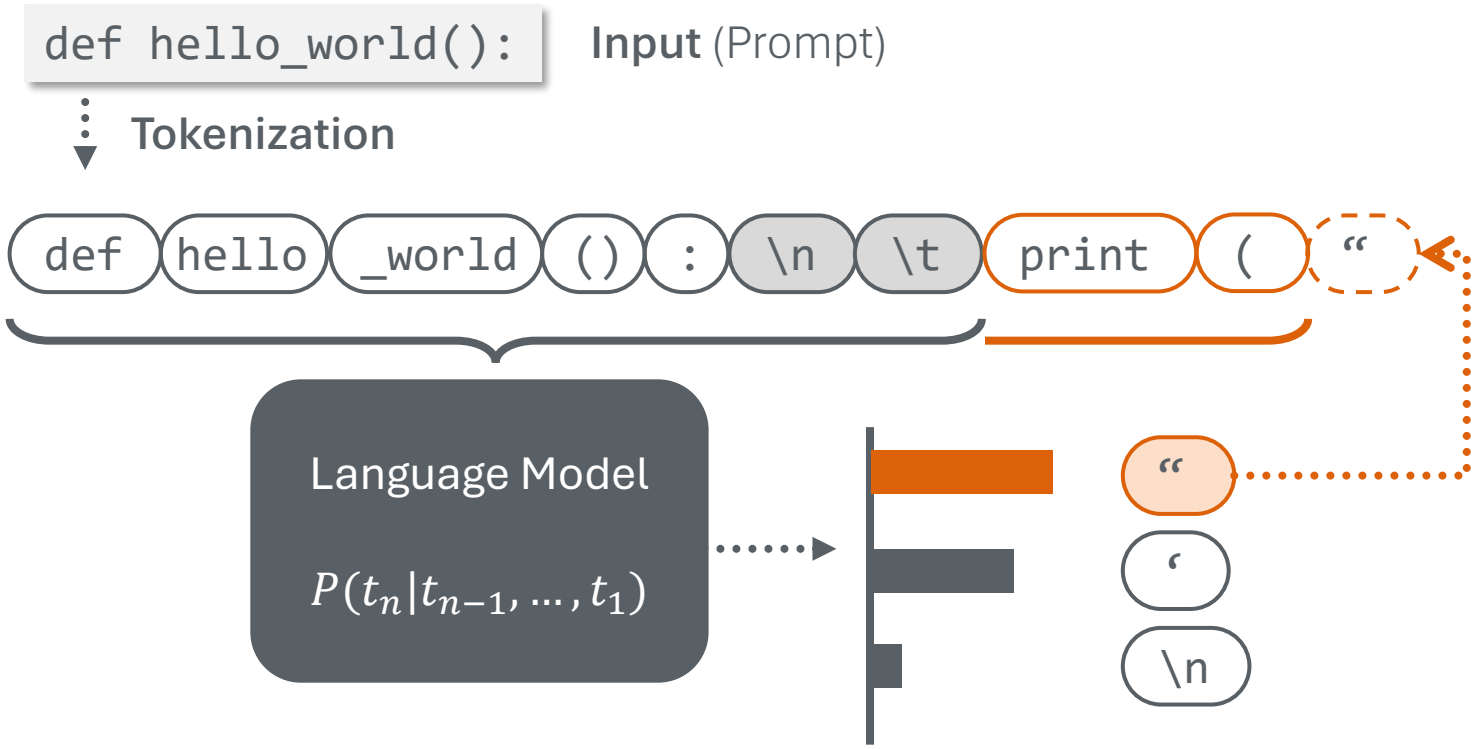
Large Language Models



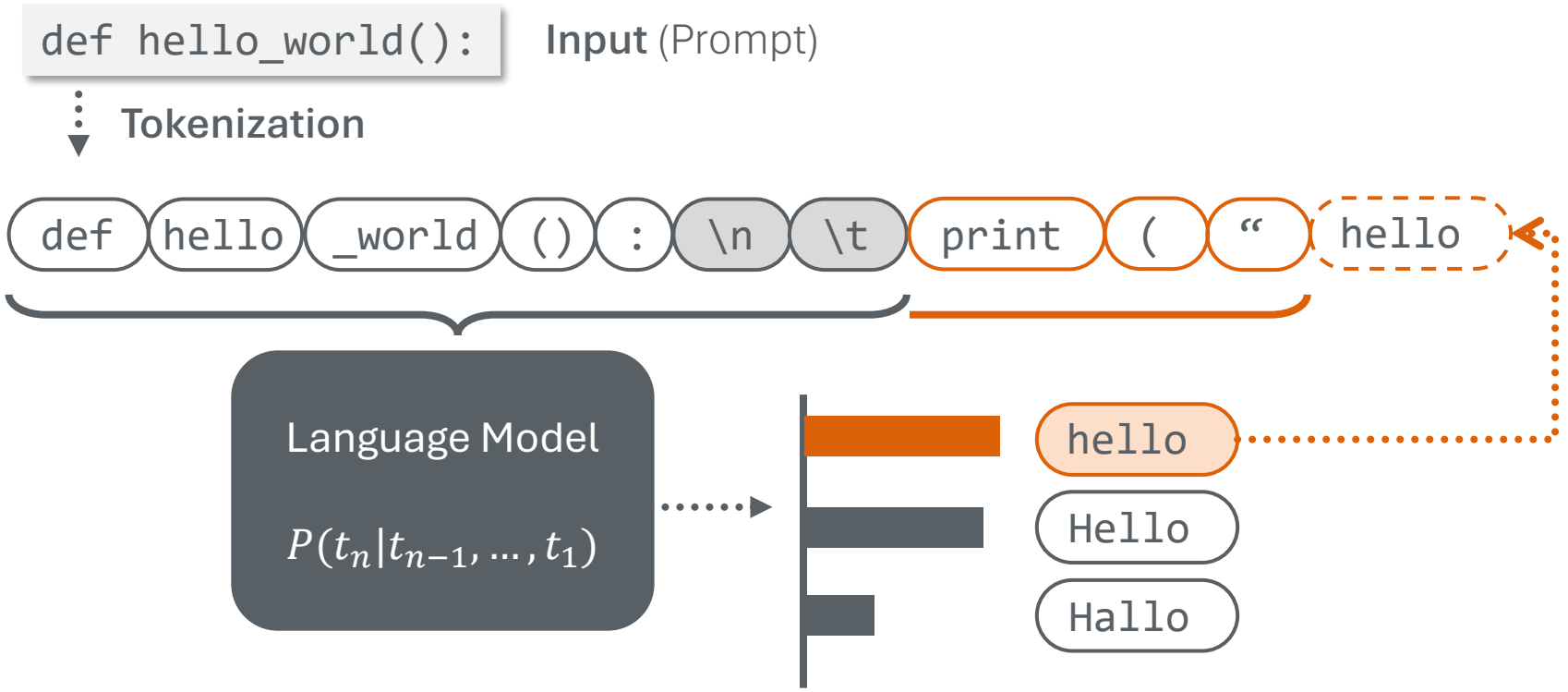
Large Language Models



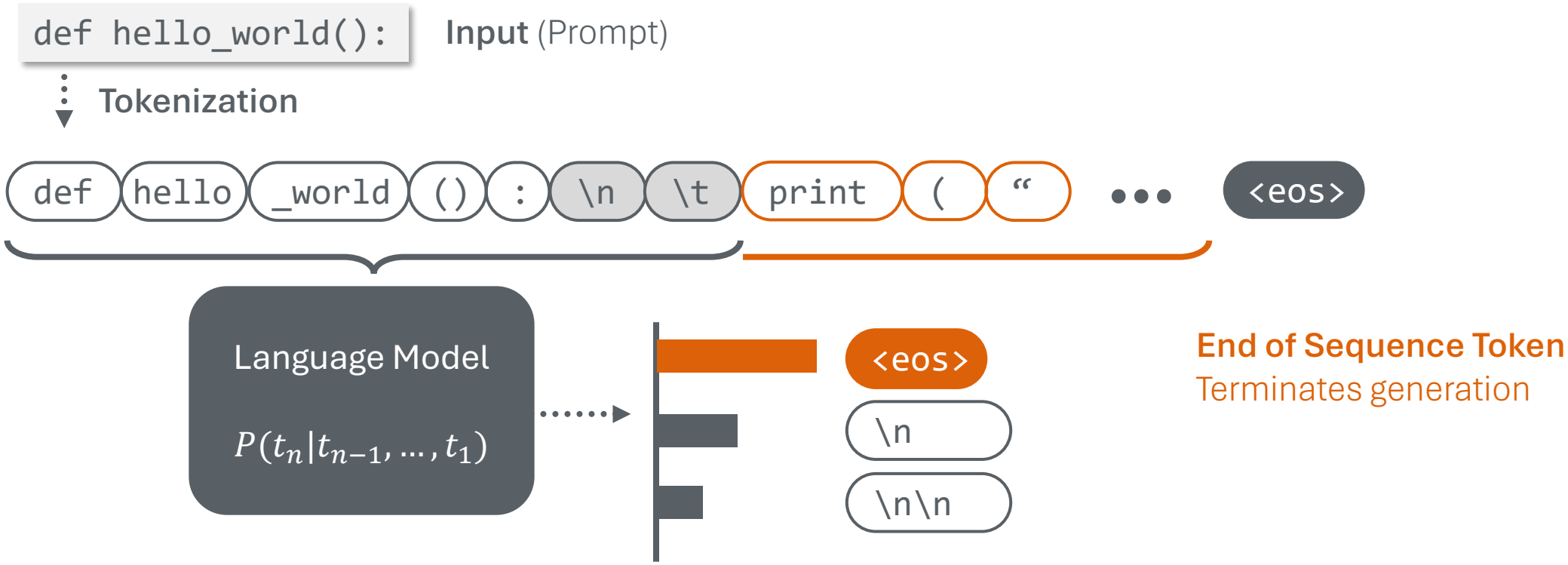
Large Language Models



Large Language Models



Large Language Models



End of Sequence Token
Terminates generation

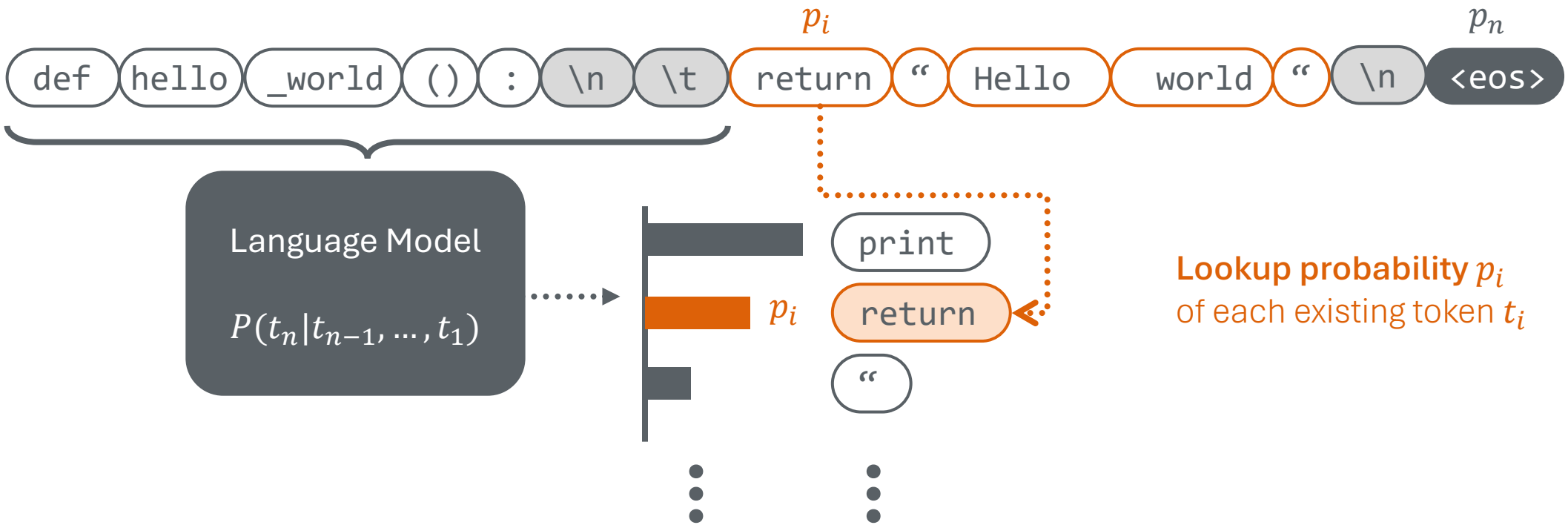
Large Language Models | **Test Prioritization**

💡 Instead of generating code, let the LLM output the probability of existing code

Large Language Models | **Test Prioritization**

$p($ **Existing Code** $) = p_1 \times \ldots \times p_n$

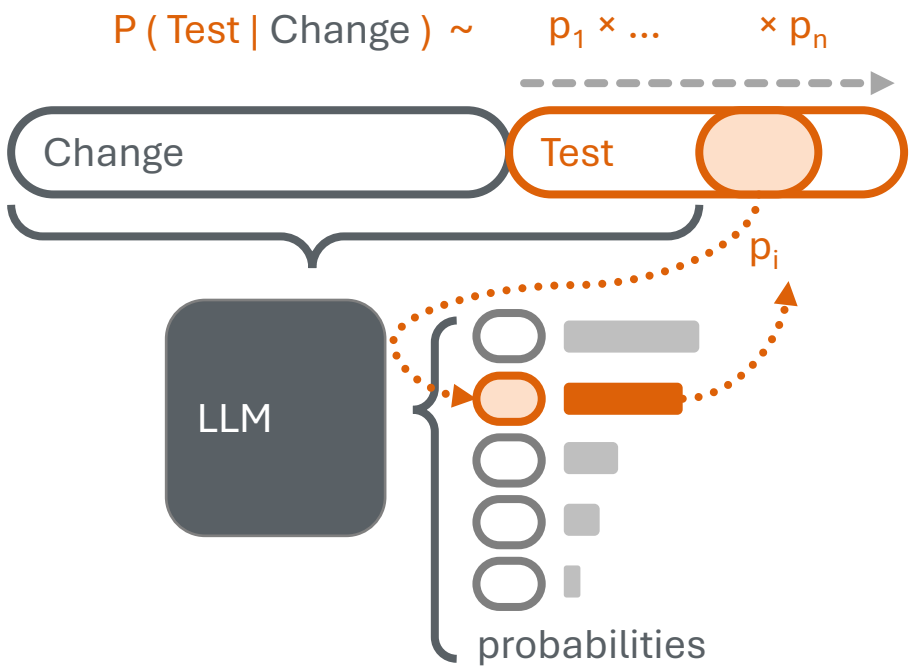
Pretend we're generating it



Lookup probability p_i
of each existing token t_i

Large Language Models | **Test Prioritization**

Instead of generating tests, let the LLM output the probability of existing tests in response to a change. Run most probable test first.



Large Language Models | **Test Prioritization**

- Format a prompt and append each test, scoring its probability
- Comment out deleted code
- Retain lexical scope

```
class AClass:

    def __init__(self):
        …

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

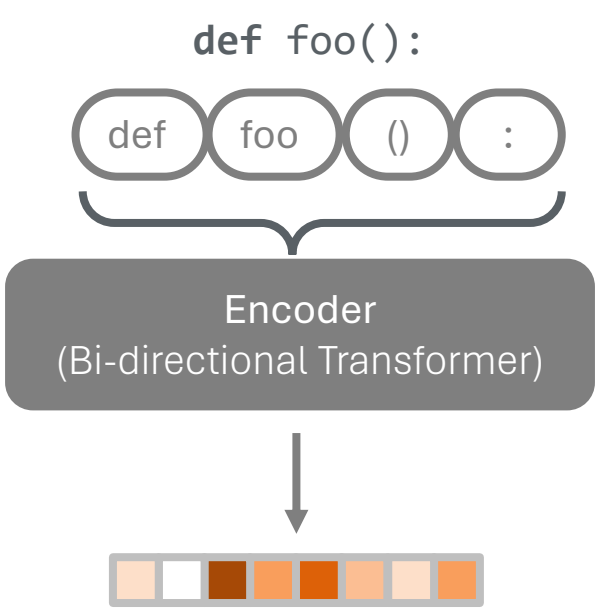
Change

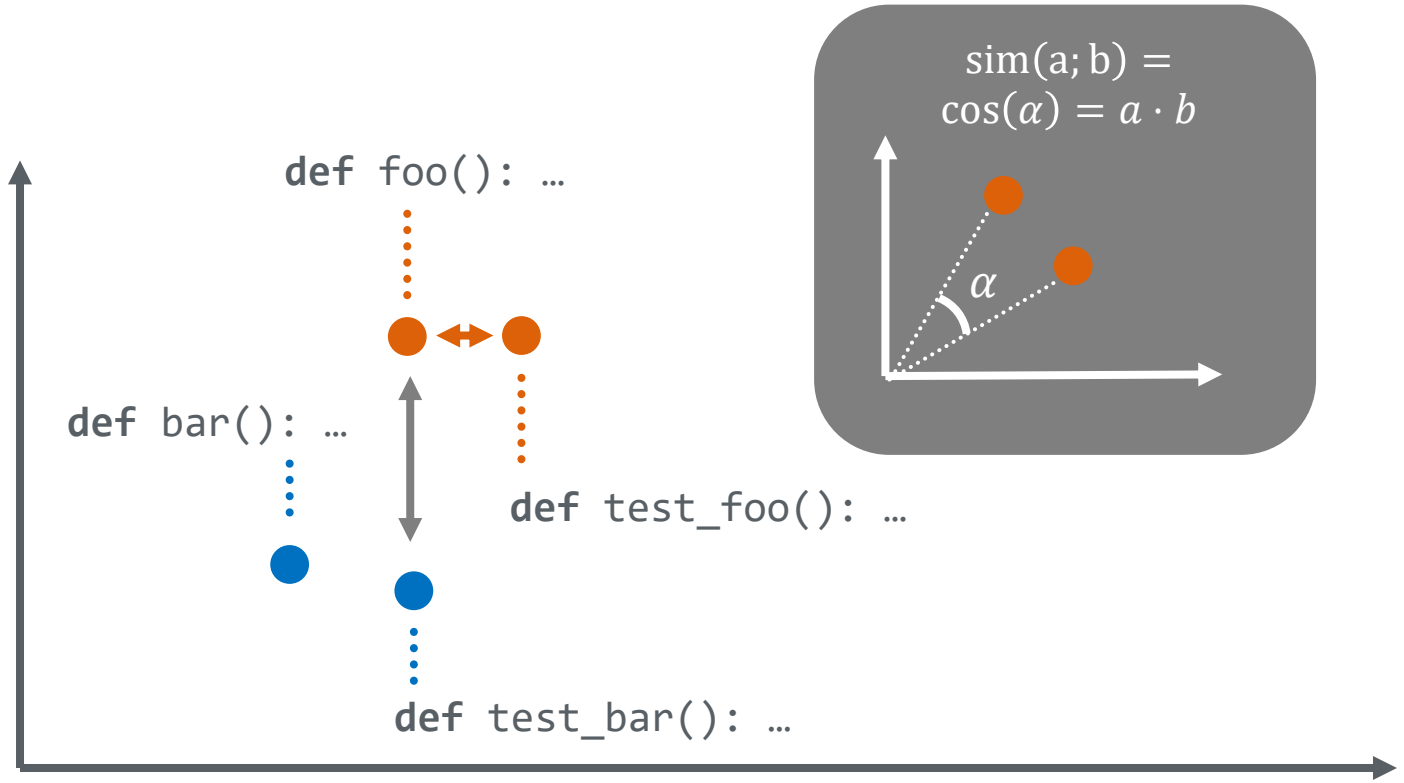
```
# /src/the_file.py
class AClass:

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

# Test validating this change:
```

LLM Prompt

Large Language Models | **Embeddings**

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





NLP- and AI-based Test Prioritization

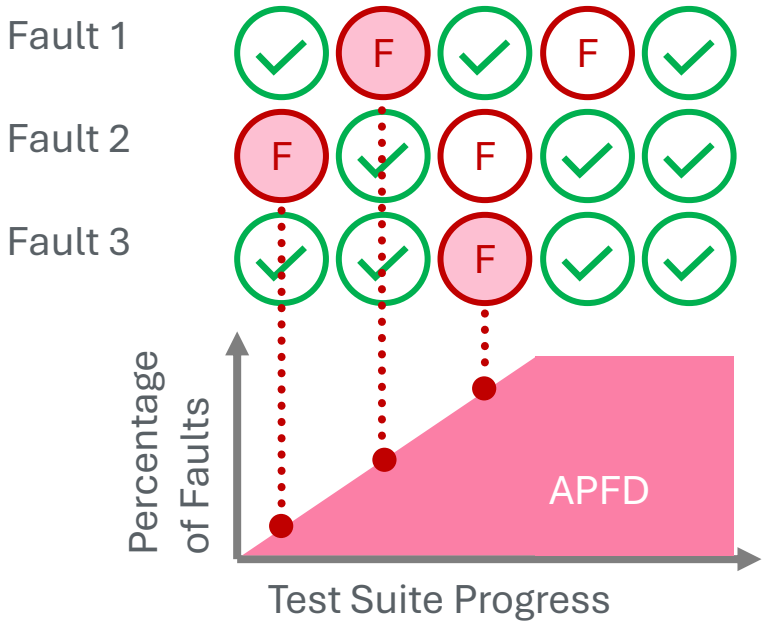
Order tests so that … run first

1. **TF-IDF:** Tests with (important) terms shared with changed code
2. **Topic Model:** Tests concerned with a similar set of topics
3. **LLM:** Tests most likely generated to test a change
4. **Embedding:** Tests semantically related to a change

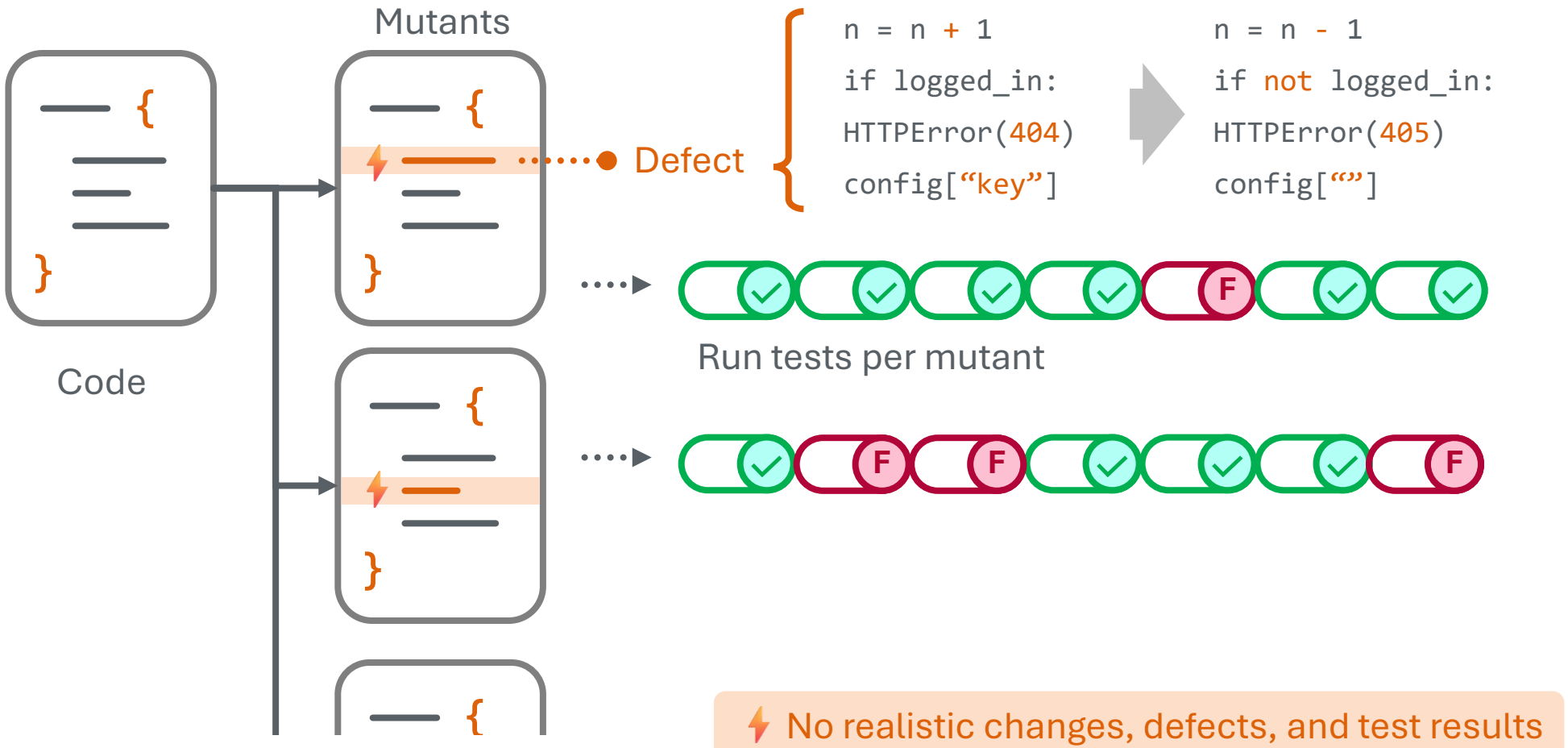
But which one is the best?

Evaluation | **Metrics**

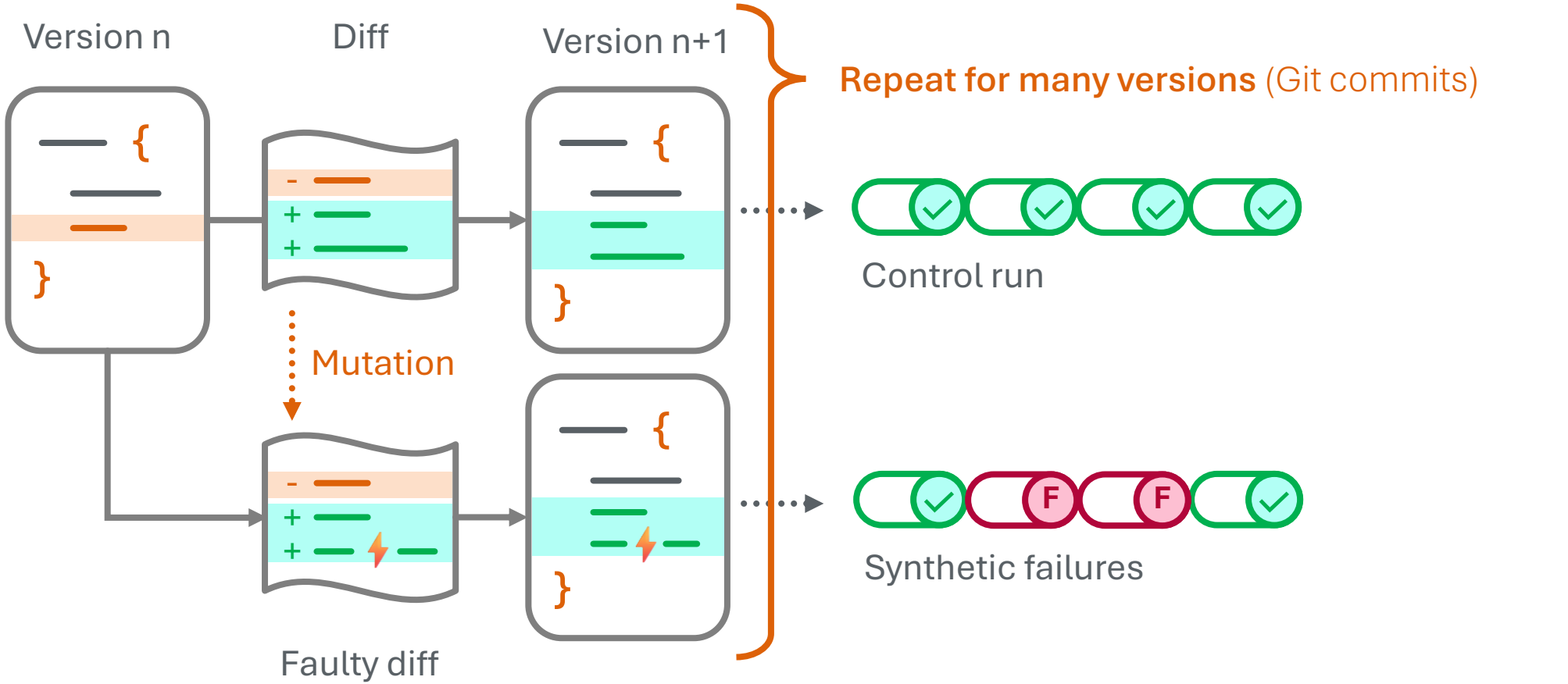
- **Performance:** average percentage of faults detected (APFD)
- Computes the area under the curve that plots the percentage of **uncovered faults so far** (y-axis) over the percentage of **already executed tests** (x-axis)



Obtaining Test Failures | **Mutation Testing**



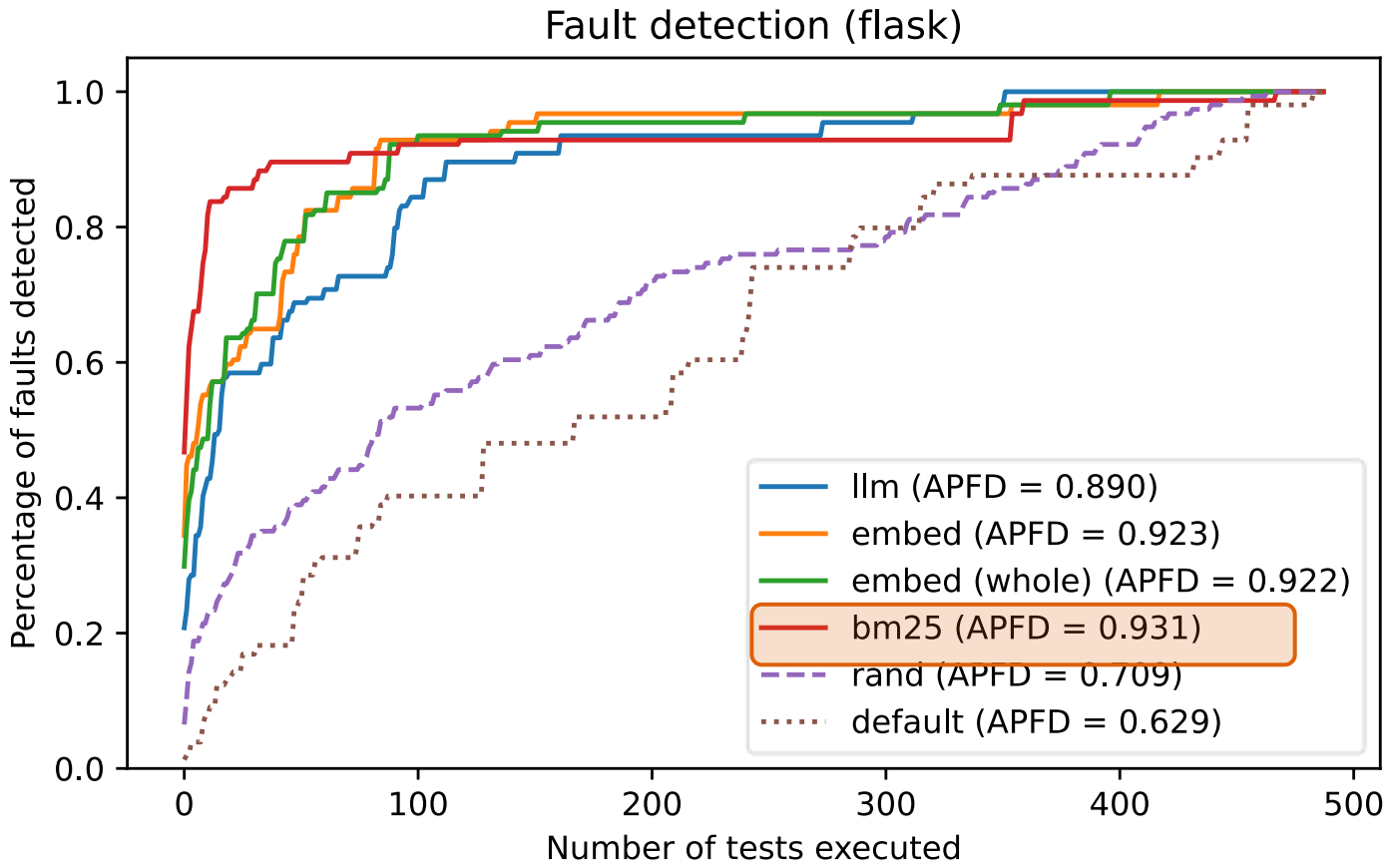
Obtaining Test Failures | **Change Mutation Testing**



✅ Fault distribution follows real changes

Results | **Example: Flask**

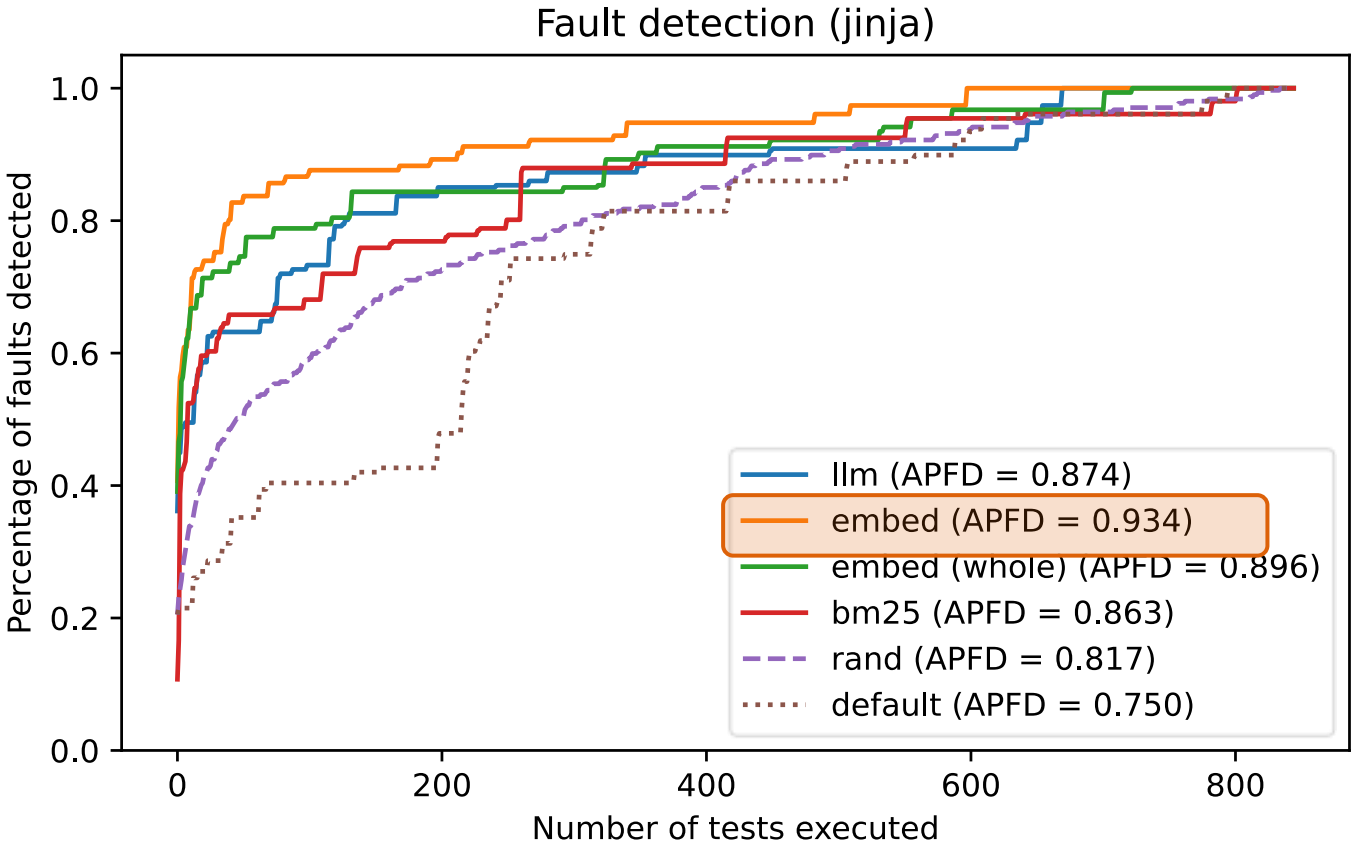
Surprise: **TF-IDF** (a very simple algorithm) sometimes beats AI



TF-IDF

Results | **Example: Jinja**

In most larger projects, an **embedding-based** test ordering is best

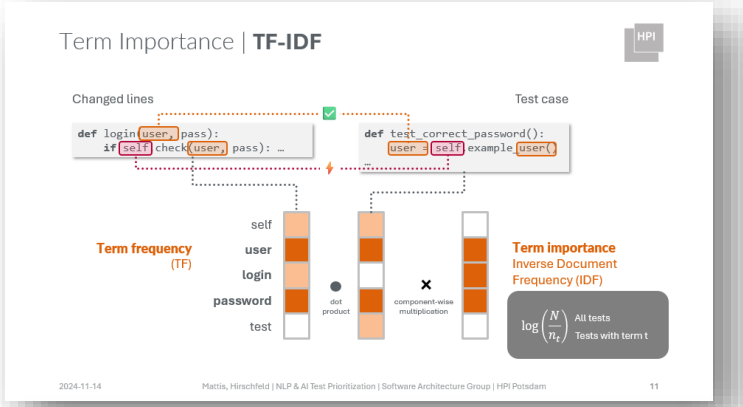


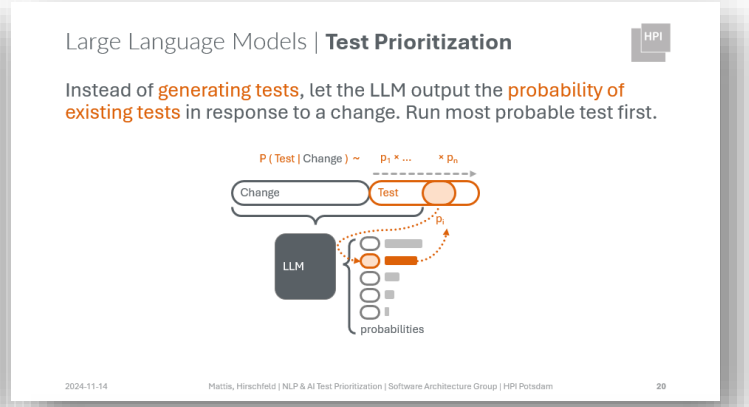
Embedding

Results | **Discussion**

- LLMs are “too nuanced” and currently too expensive
  - Penalize tests for bad practices (“Wouldn’t have written such test”)
  - **Overhead:** 10 – 20 ms GPU time > execution time of average unit test
- Often **simple heuristics** (e.g. TF-IDF) are competitive
- Limitations and Next steps
  - Models only pre-trained: **Fine-tune** LLM and embedding models on tests
  - Synthetic data: Run the experiments on real **historical data**
  - Additional features (e.g., commit messages)

Natural-language based Test Prioritization

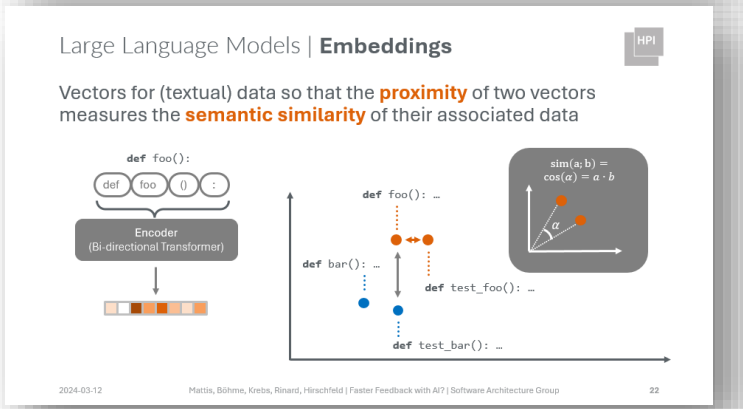


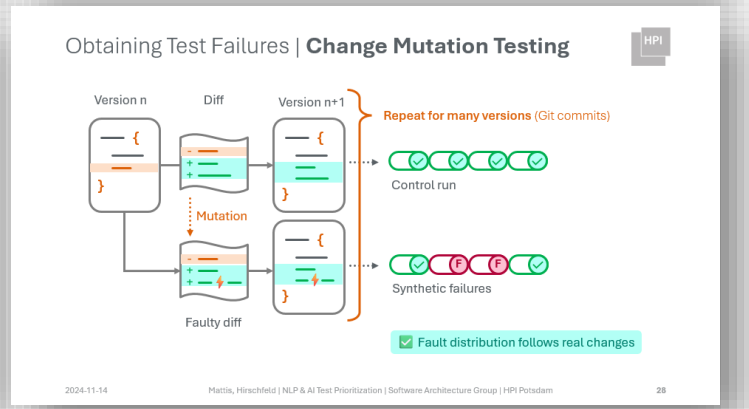


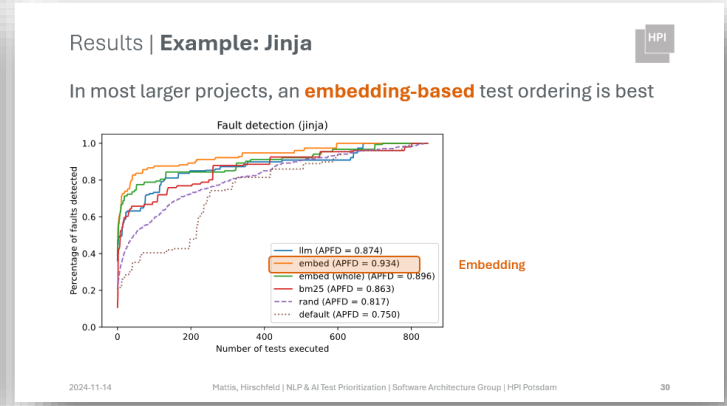
Contact:

toni.mattis@hpi.de

https://toni.mattis.berlin







Backup Slides

Selected Publications

1. T. Mattis, L. Böhme, E. Krebs, M. Rinard, R. Hirschfeld, *Faster Feedback with AI? - A Test Prioritization Study*, Programming with AI (PAI/24), Lund, Sweden, Mar. 2024
2. T. Mattis, P. Rein, F. Dürsch, and R. Hirschfeld, *RTPTorrent: An Open-source Dataset for Evaluating Regression Test Prioritization*, 17th International Conference on Mining Software Repositories (MSR), Seoul, South Korea, May 2020
3. T. Mattis and R. Hirschfeld, *Lightweight Lexical Test Prioritization for Immediate Feedback*, The Art, Science, and Engineering of Programming, Vol. 4, Nr. 3.
4. D. Meier, T. Mattis, and R. Hirschfeld, *Toward Exploratory Understanding of Software Using Test Suites*, Programming Experience (PX/21), Cambridge, UK, Mar. 2021

Conceptual Similarity | **Topic Models**

Conceptual Similarity | **Topic Models**

Distributional hypothesis: Terms that are present or absent together refer to a similar concept (topic)

Conceptual Similarity | **Topic Models**

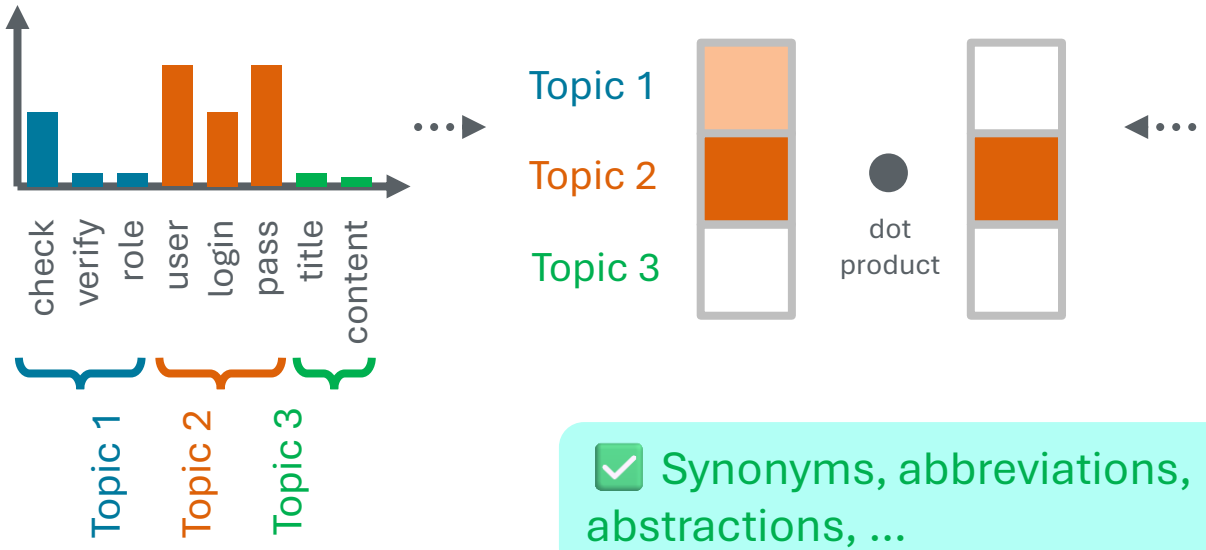
Conceptual Similarity | **Prioritizing with Topics**

Changed lines

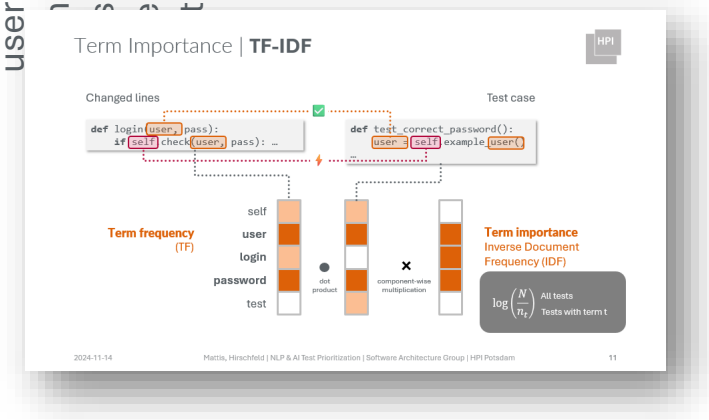
```
def login(user, pass):
    if self.check(user, pass): …
```

Test case

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



✅ Synonyms, abbreviations, abstractions, …



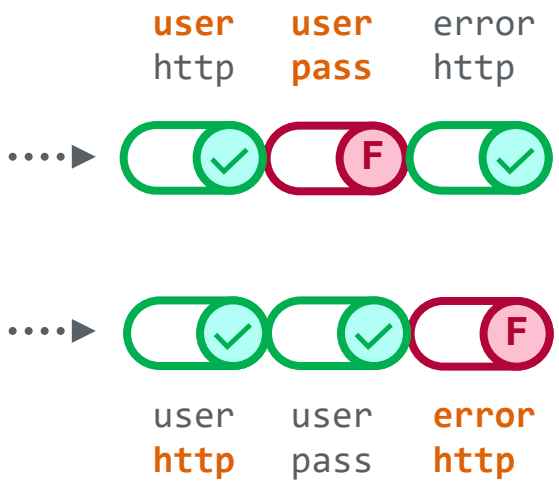
Term Importance | **Predictive Value**

Use terms that predicted past test failures

Changed lines

```
+  if user.check(pass):
```

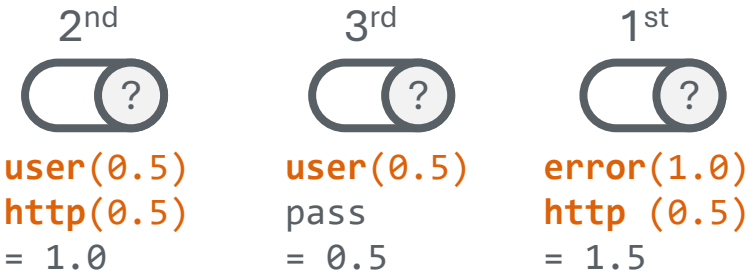
```
-  raise HTTPError(403)
```

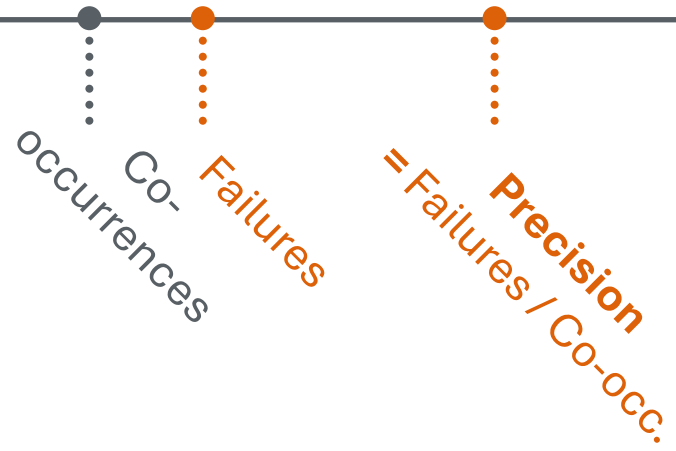


user	2	1	50%
pass	1	1	**100%**
error	1	1	**100%**
http	2	1	50%

Future change:

```
+ if user is None:
+     raise HTTPError(403)
```





Evaluation Details

Dataset: Python projects

- Well-tested
- Tests must run reproducibly many versions into the past (~10 years)
- Only include tests that pass an (un-mutated) control run

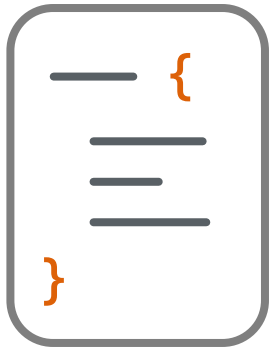
	Commits	Tests (Param.)	Faults	LOC Changed
Flask	159	390 (442)	726	12.5
Requests	43	314 (557)	188	13.8
Jinja	68	655 (829)	420	15.2

LLM-Based experiments:

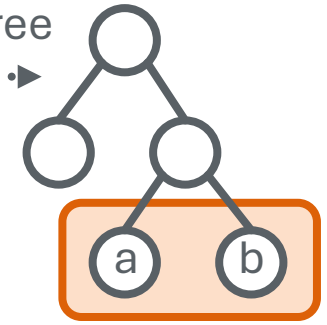
- Performed on NVidia RTX 4090 (24GB GPU Memory)
- Models: CodeLlama-7B (initial), StableCode-3B (in the paper), CodeGemma-1.1-2b (now)

Bi-term Topic Model for Code

- Traditional topic models (e.g., LDA) designed for documents
- Code has a smaller vocabulary than natural language documents, less redundancy, and a hierarchy/graph-like structure (no apparent document boundaries)



Syntax Tree



Co-occurrence relation

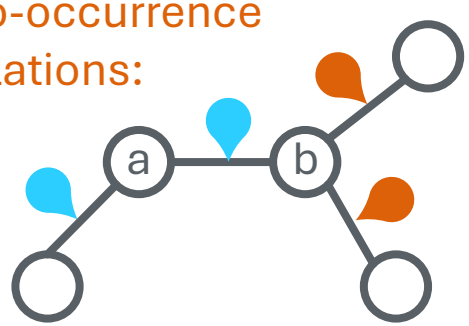
Probability of names co-occurring

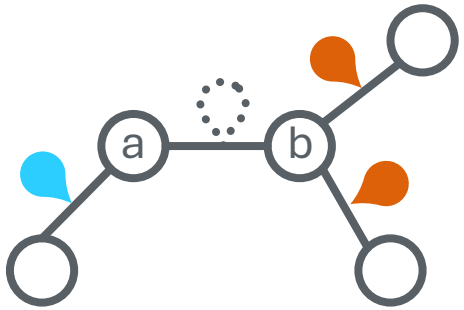
Probability of names belonging to concept/topic c

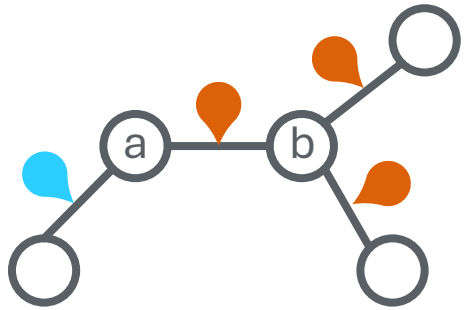
$$P(a, b|c) = p(a|c)p(b|c)$$

Fitting a Code Topic Model via Edge Clustering

Co-occurrence relations:







⚠ **Multigraph** (many edges per pair of nodes)

1. Randomly color edges
2. Re-sample edge color from adjacent edges
3. Assign color, repeat 2. & 3. for each edge

Gibbs sampling (Monte Carlo method)

$$p(c|a,b) = \begin{cases} \frac{1+\alpha}{2+2\alpha} & c = \text{🔵} \\ \frac{2+\alpha}{2+2\alpha} & c = \text{🟠} \end{cases}$$

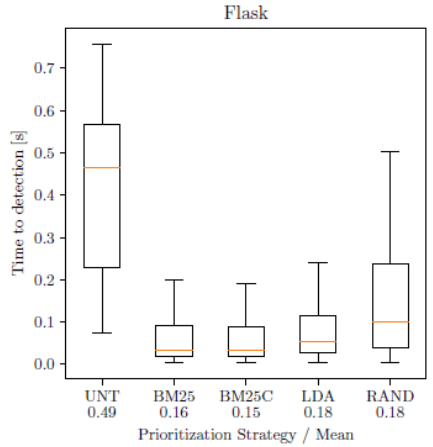
Smoothing term (Dirichlet prior)

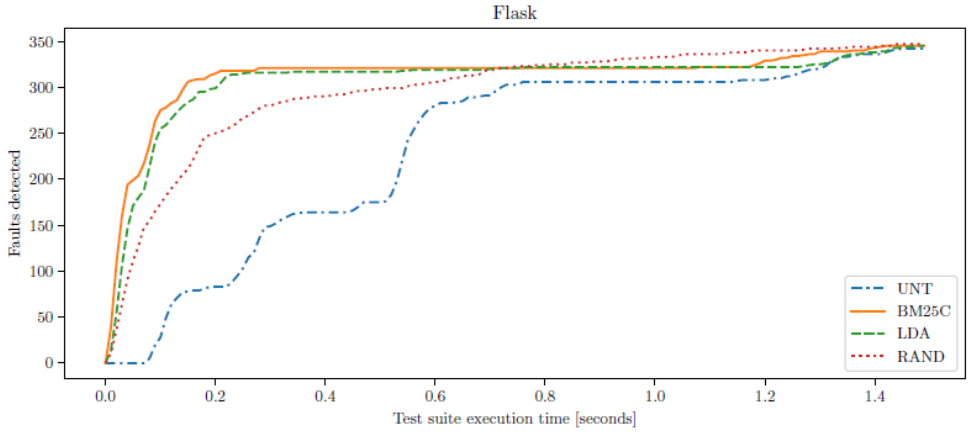
Iterate until Convergence. Then:
Topics = colors

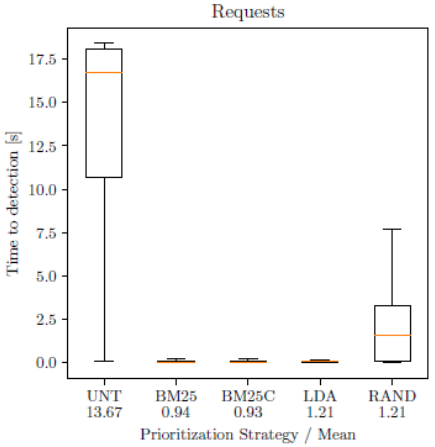
$$p(a|c) = \frac{e_{a,c}}{e_c}$$

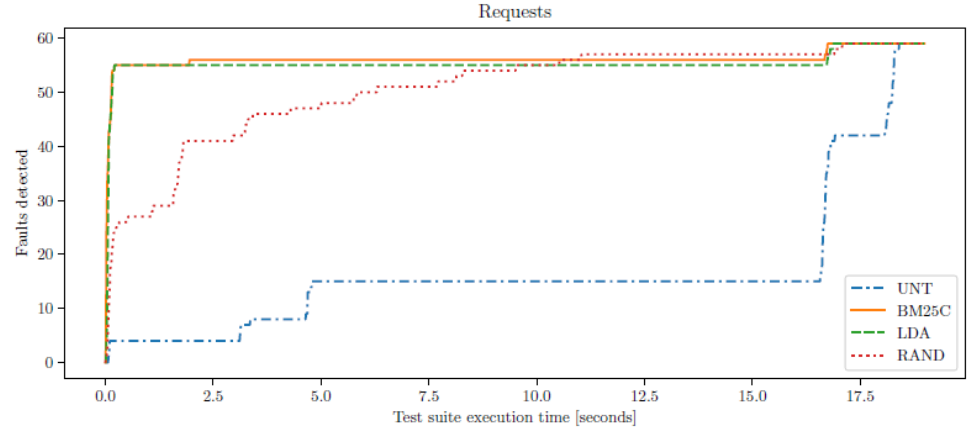
Edges from a colored c

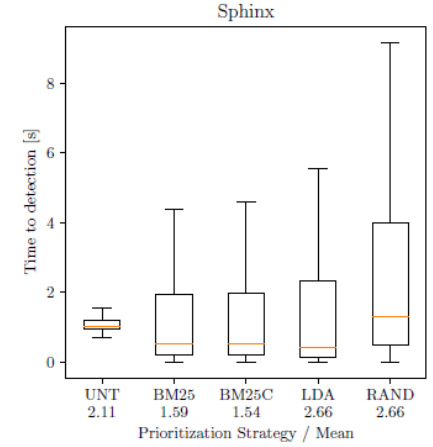
Edges from a

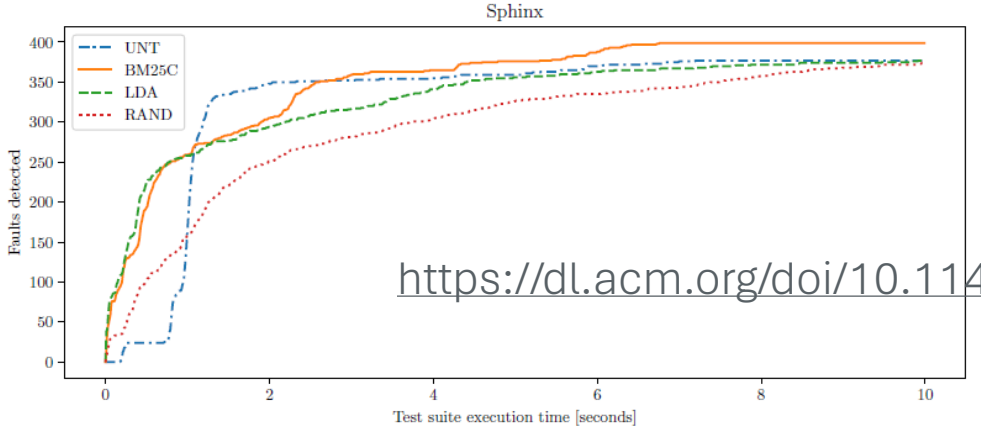




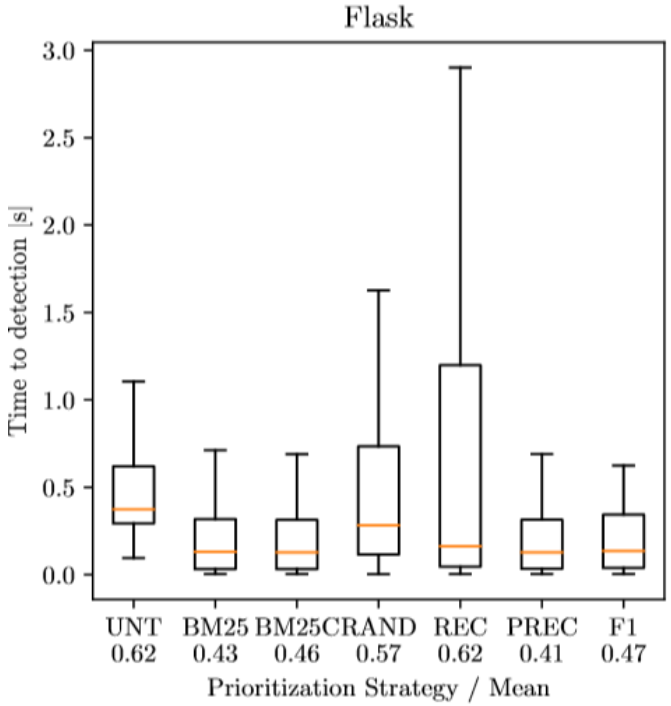


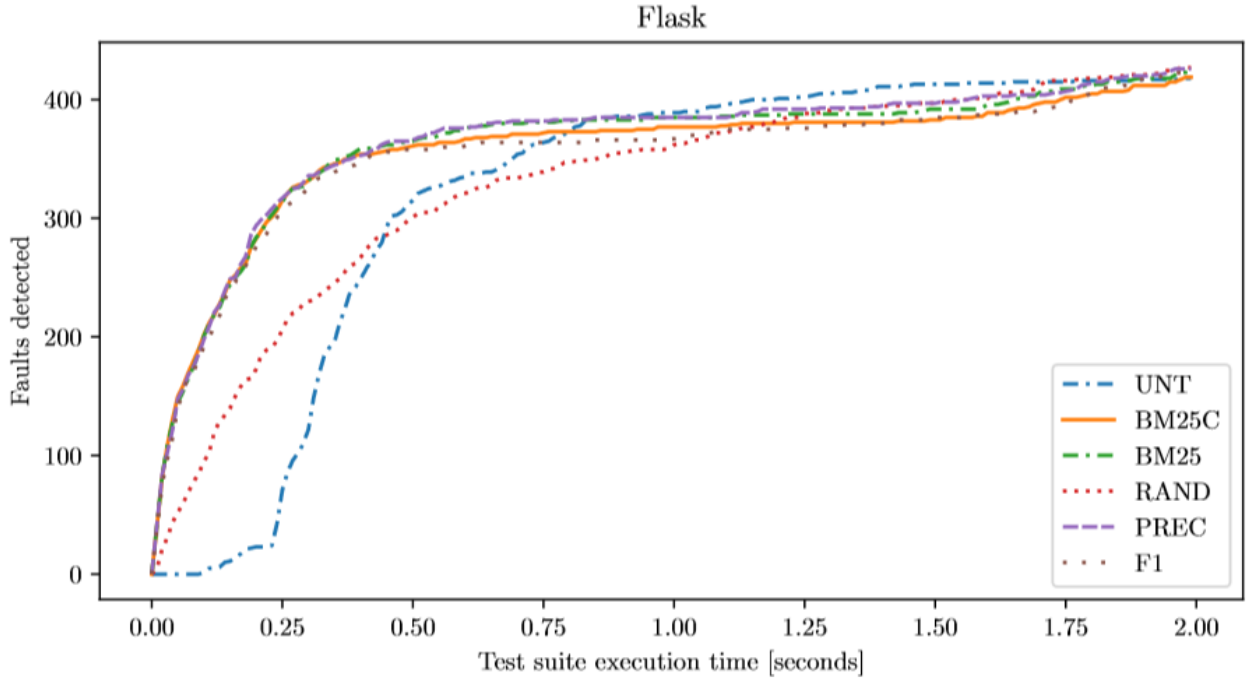


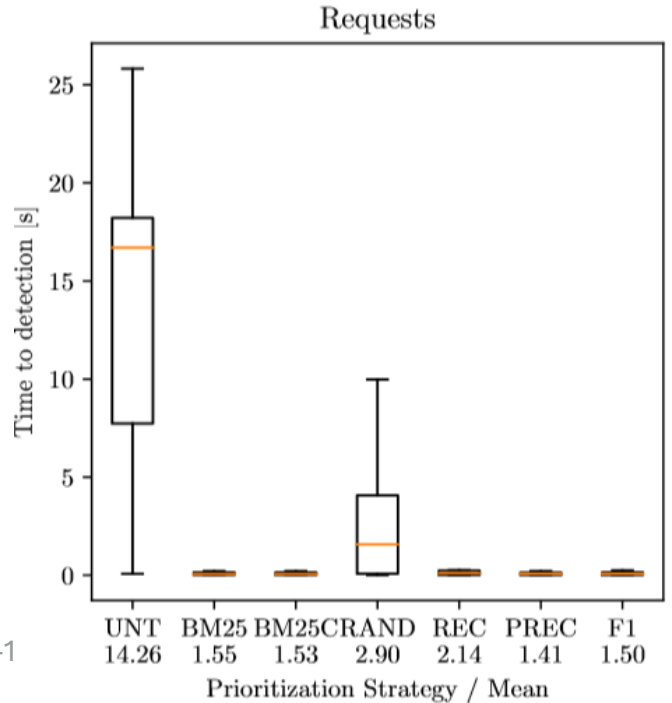


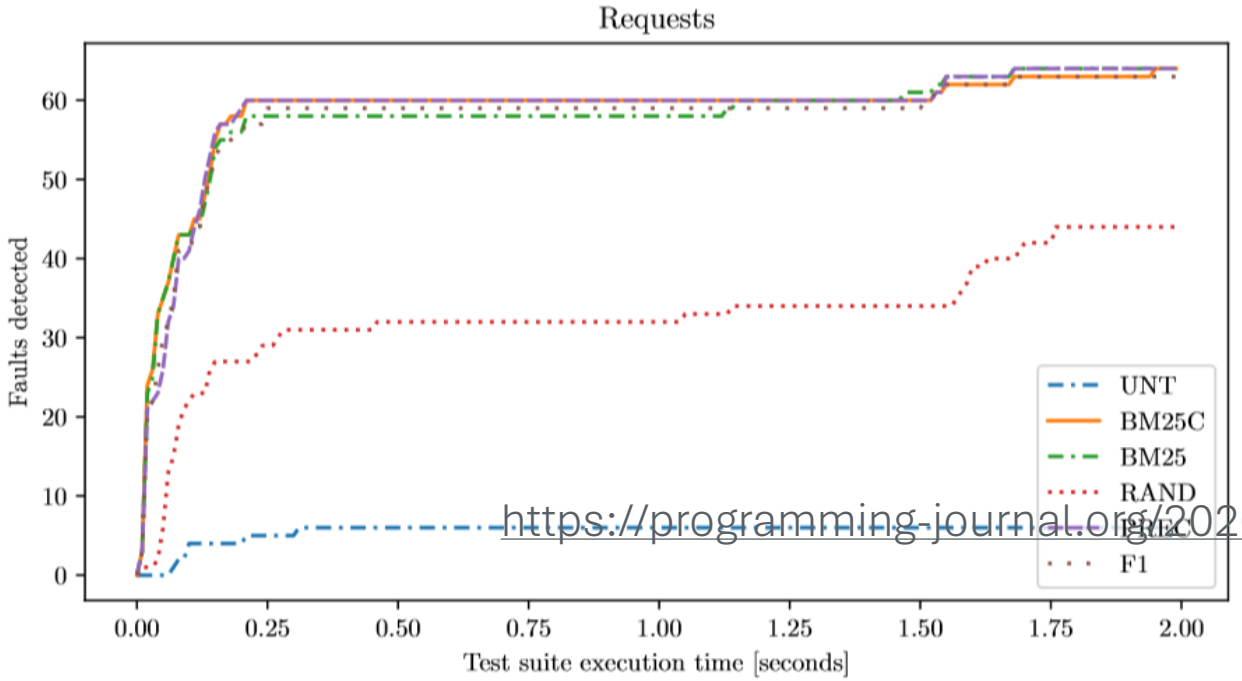


https://dl.acm.org/doi/10.1145/3328433.3328455

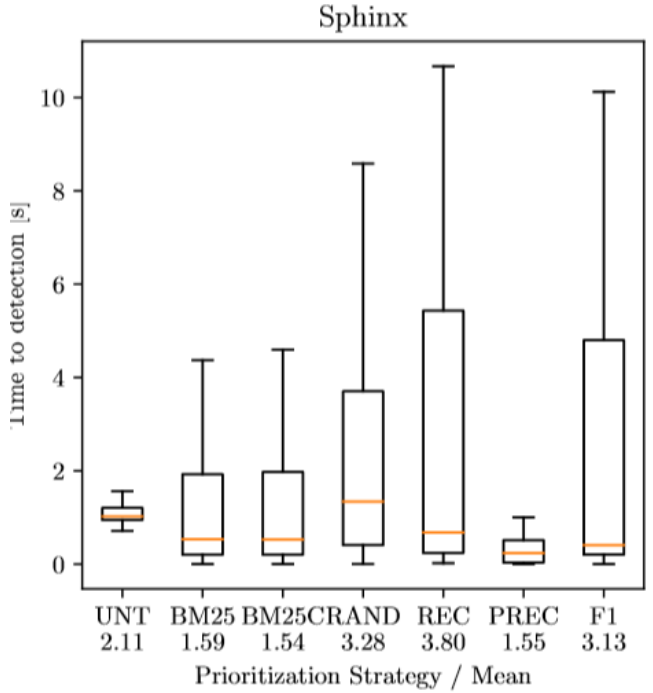


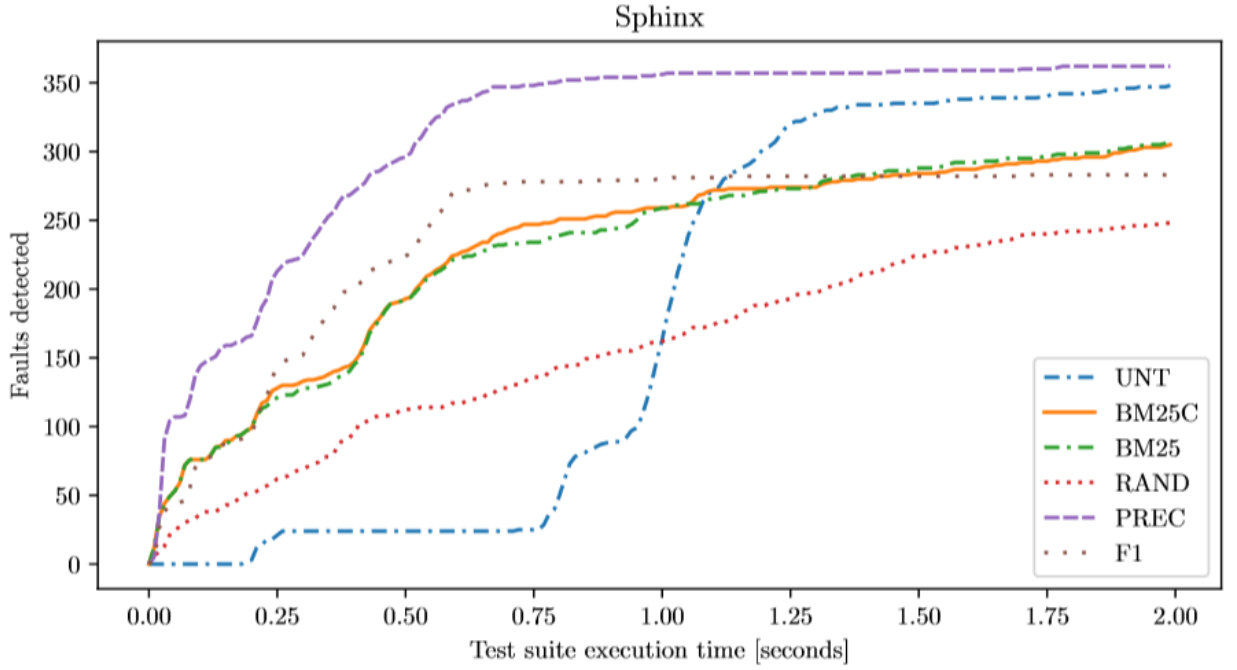


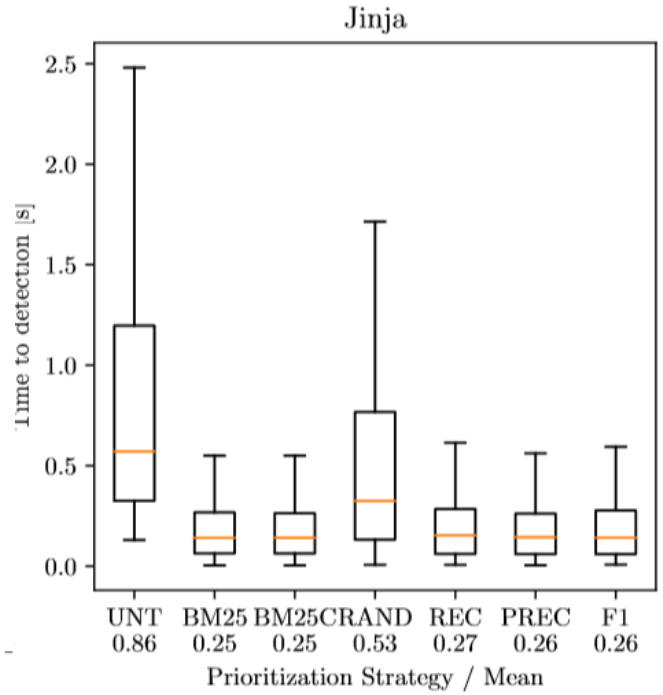


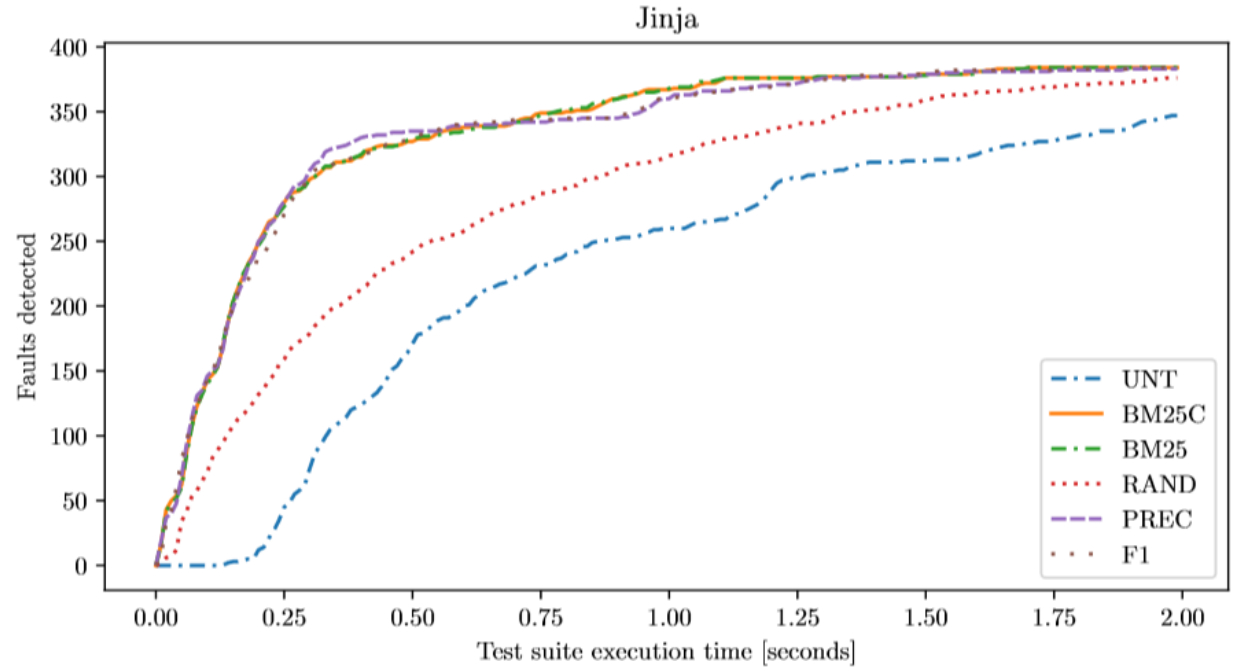


https://programming-journal.org/2020/4/12/


Sphinx
Time to detection [s]
0
2
4
6
8
10
UNT 2.11
BM25 1.59
BM25C 1.54
RAND 3.28
REC 3.80
PREC 1.55
F1 3.13
Prioritization Strategy / Mean


Sphinx
Faults detected
0
50
100
150
200
250
300
350
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Test suite execution time [seconds]
UNT
BM25C
BM25
RAND
PREC
F1


Jinja
Time to detection [s]
0.0
0.5
1.0
1.5
2.0
2.5
UNT 0.86
BM25 0.25
BM25C 0.25
RAND 0.53
REC 0.27
PREC 0.26
F1 0.26
Prioritization Strategy / Mean


Jinja
Faults detected
0
50
100
150
200
250
300
350
400
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Test suite execution time [seconds]
UNT
BM25C
BM25
RAND
PREC
F1