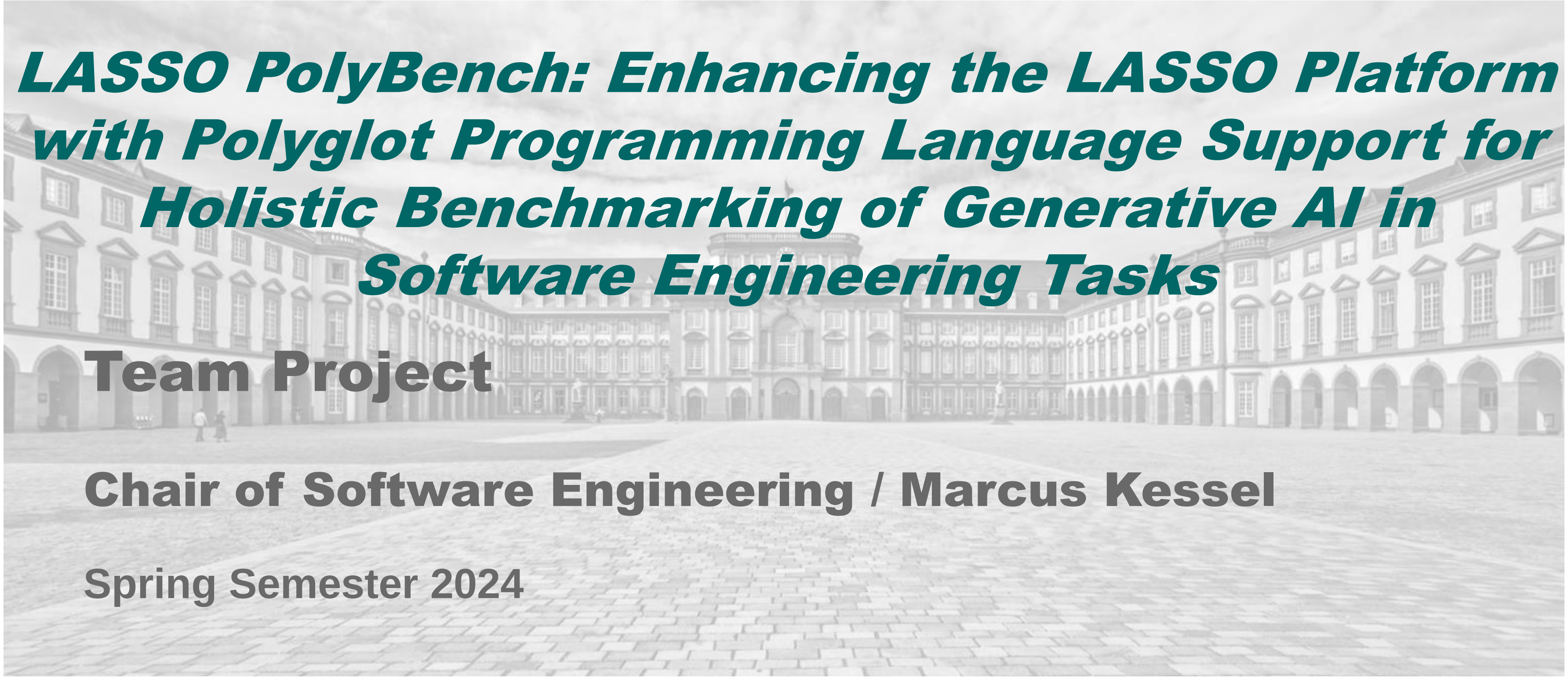


LASSO PolyBench: Enhancing the LASSO Platform with Polyglot Programming Language Support for Holistic Benchmarking of Generative AI in Software Engineering Tasks

Team Project

Chair of Software Engineering / Marcus Kessel

Spring Semester 2024

UNIVERSITY
OF MANNHEIM
School of Business Informatics
and Mathematics

Generative AI for Software Engineering Tasks

LLMs (chatbots) trained on massive amounts of open source code

OpenAI

ChatGPT

Codex

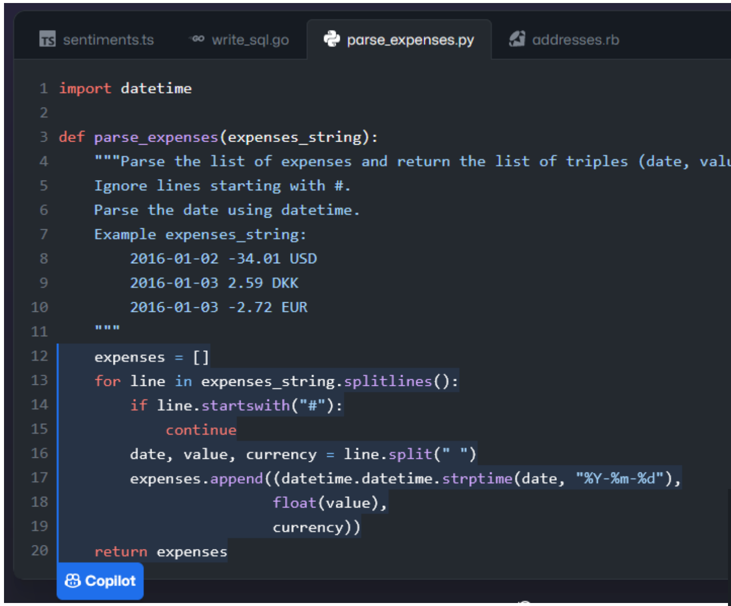
GitHub Copilot

tabnine

StarCoder

… many more

Code & Test Generation Tasks

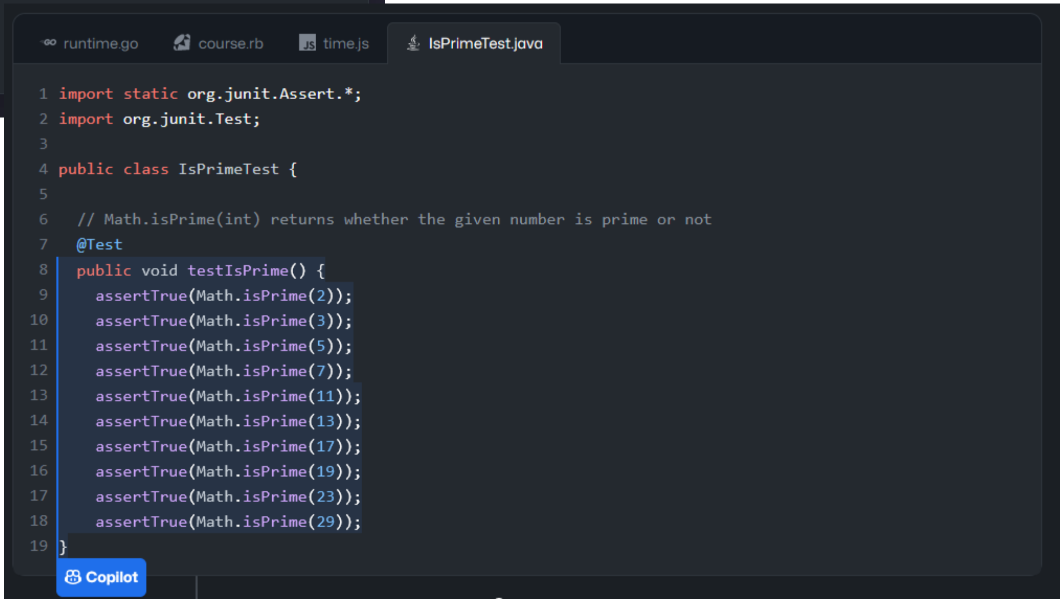


```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                        float(value),
                        currency))
    return expenses
```

Code Generation
(Program Synthesis)

GitHub Copilot



```java
import static org.junit.Assert.*;
import org.junit.Test;

public class IsPrimeTest {

  // Math.isPrime(int) returns whether the given number is prime or not
  @Test
  public void testIsPrime() {
    assertTrue(Math.isPrime(2));
    assertTrue(Math.isPrime(3));
    assertTrue(Math.isPrime(5));
    assertTrue(Math.isPrime(7));
    assertTrue(Math.isPrime(11));
    assertTrue(Math.isPrime(13));
    assertTrue(Math.isPrime(17));
    assertTrue(Math.isPrime(19));
    assertTrue(Math.isPrime(23));
    assertTrue(Math.isPrime(29));
}
```

Test Generation
(Inputs and Outputs)

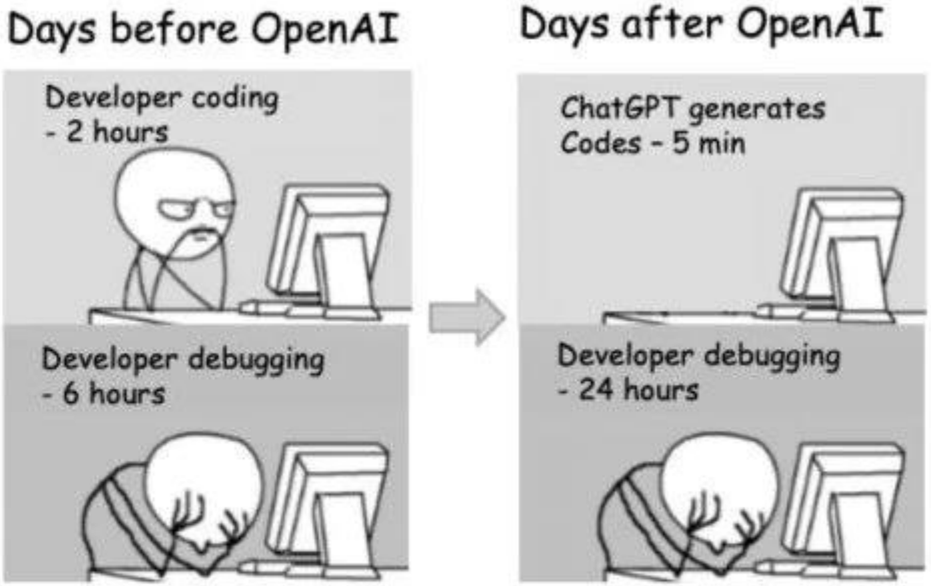
Jack of all Trades, Master of None (?)

“… the robots are coming …”

“… they replace developers …”

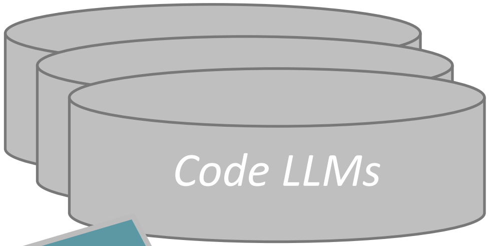
*“… hallucinating bullsh** …”*

“… frees from boring tasks …”


Days before OpenAI
Developer coding - 2 hours
Developer debugging - 6 hours
Days after OpenAI
ChatGPT generates Codes - 5 min
Developer debugging - 24 hours

/r/ProgrammerHumor

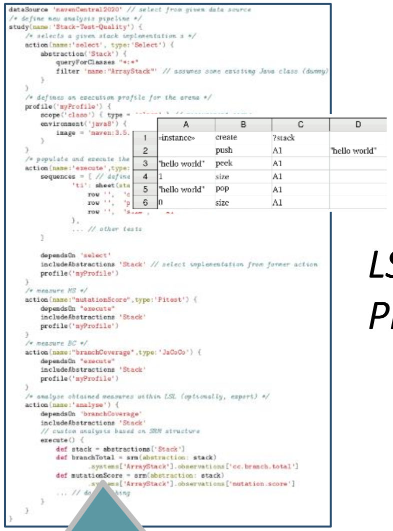
LASSO PolyBench



Measure and Analyze Functional/Non-Functional Behaviour

Execution-based Program Analysis at Scale

<u>*Replicate State-of-the-Art Benchmarks*</u>
Curated Coding Task Collections
→ *HumanEval, MBPP, MultiPL-E* etc.



```
dataSource 'mavenCentral2020' // select from given data source
/* define new analysis pipeline */
study(name:'Stack-Test-Quality') {
    /* selects a given stack implementation s */
    action(name:'select', type:'Select') {
        abstraction('Stack') {
            queryForClasses "*:*"
            filter 'name:"ArrayStack"' // assumes some existing Java class (dummy)
        }
    }
    /* defines an execution profile for the arena */
    profile('myProfile') {
        scope('class') { type =
        environment('java8') {
            image = 'maven:3.5.
        }
    }
    /* populate and execute the
    action(name:'execute',type:
        sequences = [ // define
                'ti': sheet(sta
                    row '',  'c
                    row '',  'p
                    row '',  'size',
                },
                ... // other tests
        ]

        dependsOn 'select'
        includeAbstractions 'Stack' // select implementation from former action
        profile('myProfile')
    }
    /* measure MS */
    action(name:"mutationScore",type:'Pitest') {
        dependsOn "execute"
        includeAbstractions 'Stack'
        profile('myProfile')
    }
    /* measure BC */
    action(name:"branchCoverage",type:'JaCoCo') {
        dependsOn "execute"
        includeAbstractions 'Stack'
        profile('myProfile')
    }
    /* analyse obtained measures within LSL (optionally, export) */
    action(name:'analyse') {
        dependsOn 'branchCoverage'
        includeAbstractions 'Stack'
        // custom analysis based on SRM structure
        execute() {
            def stack = abstractions['Stack']
            def branchTotal = srm(abstraction: stack)
                    .systems['ArrayStack'].observations['cc.branch.total']
            def mutationScore = srm(abstraction: stack)
                    .sy   ms['ArrayStack'].observations['mutation.score']
            ... // do     hing
        }
    }
}
```

	A	B	C	D
1	«instance»	create	?stack	
2		push	A1	"hello world"
3	"hello world"	peek	A1	
4	1	size	A1	
5	"hello world"	pop	A1	
6	0	size	A1	

LSL Analysis Pipelines

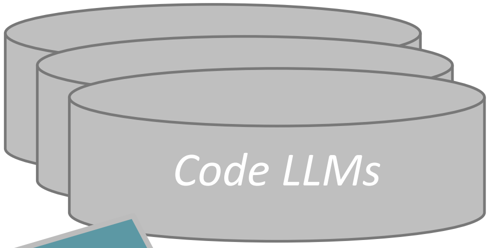
Functionally correct?
Quality of the code?

LASSO PolyBench

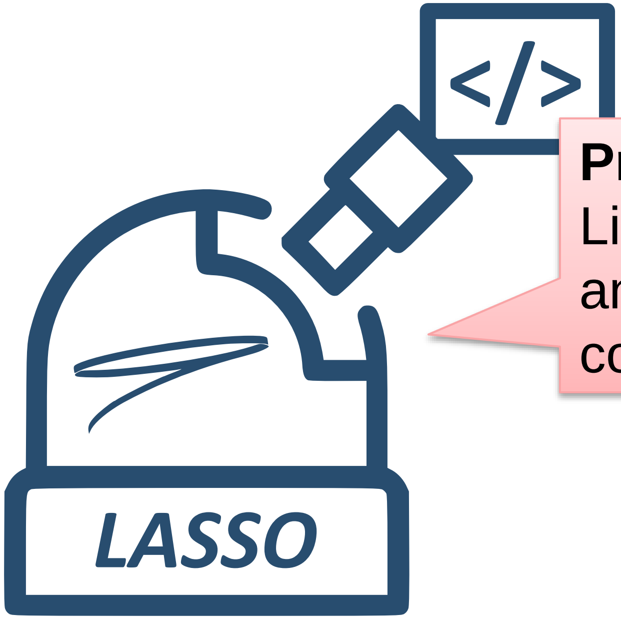
<u>*Replicate State-of-the-Art Benchmarks*</u>

Curated Coding Task Collections

→ *HumanEval, MBPP, MultiPL-E* etc.

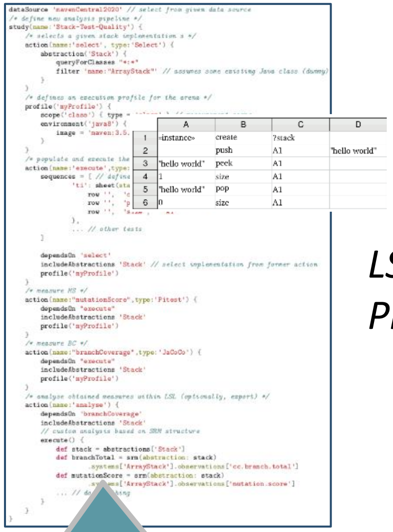


Measure and Analyze Functional/Non-Functional Behaviour



Problem
Limited to the analysis of <u>Java</u> code (problems)

Execution-based Program Analysis at Scale



```
dataSource 'mavenCentral2020' // select from given data source
/* define new analysis pipeline */
study(name:'Stack-Test-Quality') {
    /* selects a given stack implementation s */
    action(name:'select', type:'Select') {
        abstraction('Stack') {
            queryForClasses "*:*"
            filter 'name:"ArrayStack"' // assumes some existing Java class (dummy)
        }
    }
    /* defines an execution profile for the arena */
    profile('myProfile') {
        scope('class') { type =
        environment('java8') {
            image = 'maven:3.5.
        }
    }
    /* populate and execute the
    action(name:'execute',type:
        sequences = [ // define
                't1': sheet(sta
                    row '',  'c
                    row '',  'p
                    row '',  'size',
                },
                ... // other tests
        ]

        dependsOn 'select'
        includeAbstractions 'Stack' // select implementation from former action
        profile('myProfile')
    }
    /* measure MS */
    action(name:"mutationScore",type:'Pitest') {
        dependsOn "execute"
        includeAbstractions 'Stack'
        profile('myProfile')
    }
    /* measure BC */
    action(name:"branchCoverage",type:'JaCoCo') {
        dependsOn "execute"
        includeAbstractions 'Stack'
        profile('myProfile')
    }
    /* analyse obtained measures within LSL (optionally, export) */
    action(name:'analyse') {
        dependsOn 'branchCoverage'
        includeAbstractions 'Stack'
        // custom analysis based on SRM structure
        execute() {
            def stack = abstractions['Stack']
            def branchTotal = srm(abstraction: stack)
                    .systems['ArrayStack'].observations['cc.branch.total']
            def mutationScore = srm(abstraction: stack)
                    .systems['ArrayStack'].observations['mutation.score']
            ... // do
        }
    }
}
```

	A	B	C	D
1	«instance»	create	?stack	
2		push	A1	"hello world"
3	"hello world"	peek	A1	
4	1	size	A1	
5	"hello world"	pop	A1	
6	0	size	A1	

LSL Analysis Pipelines

Functionally correct?
Quality of the code?

Goal (1)

- LASSO is a leading edge software observatorium that allows advanced search and analysis techniques to be applied to “big code”. Among other things, this simplifies experimentation and the validation of tools and software engineering approaches.
- LASSO is currently limited to Java programming language support
- Goals that support the creation of LASSO PolyBench
  - add polyglot programming language support to the LASSO platform by starting with **Python language support**. This includes
    - crawling, parsing and indexing Python code
    - creating a test driver (LASSO Arena) to execute and analyze Python code of interest
  - demonstrate the **replication of benchmarks for Python coding problems** using LASSO’s concepts and data structures
    - setting up analysis pipelines in LASSO’s scripting language, LSL, to automate the experimentation process
  - demonstrate **cross-checking of code properties** (i.e., behaviour, quality etc.) across two programming languages (Java and Python)

Goal (2)

- Participants
  - 4-6 students
- Length
  - 6 months
- Prerequisites
  - Python and/or Java programming skills
  - Basic understanding of machine learning
- Language
  - English
- Organisation
  - Goals and timetable defined by agreement with the supervisor
- Applicable to MMDS: yes
- Online: By agreement

- Supervisor
  - Marcus Kessel

Open Positions – Student Assistants (HiWis)

- Be part of **AUTOR – Automated Test Oracle Recommendation**
  - Funding provided by the Ministerium für Wissenschaft, Forschung and Kunst Baden-Württemberg through *Research Seed Capital (RiSC)*
- A research project at the intersection of
  - *software engineering* (big code, mining software repositories, software testing),
  - *data science* (machine learning techniques, AI models …)
- Are you interested in … ?
  - Software testing and empirical software engineering
  - State-of-the-art recommendation and generation techniques for coding tasks
- Send an (informal) email to Marcus Kessel (marcus.kessel@uni-mannheim.de)