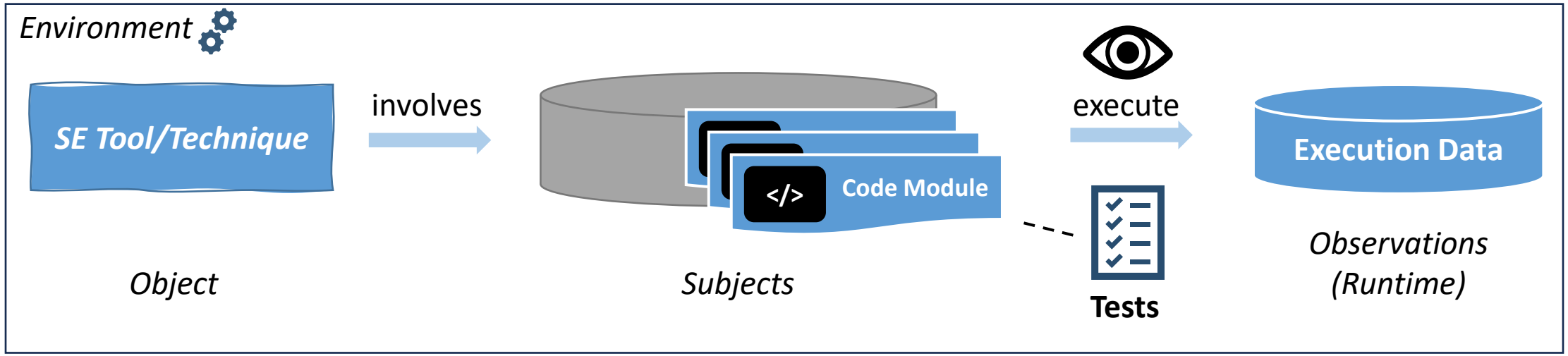
# Promoting Open Science in Test-driven Software Experiments

UNIVERSITÄT MANNHEIM

Marcus Kessel marcus.kessel@uni-mannheim.de, Colin Atkinson colin.atkinson@uni-mannheim.de

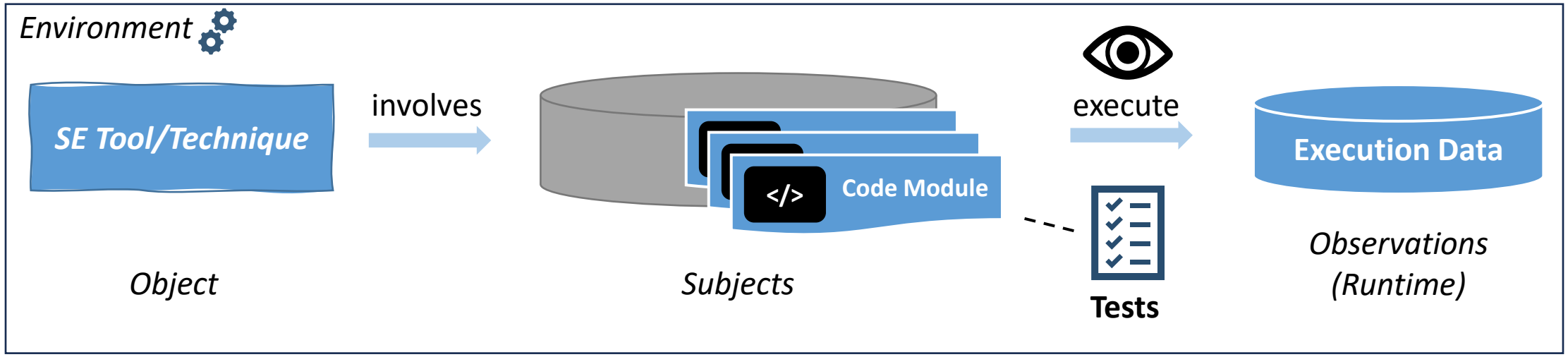
*ICST 2025 @ Naples, Italy – Journal First (Journal of Systems and Software)*

# Test-Driven Software Experiments (TDSEs)




→ TDSEs are experiments that involve controlled testing of software subjects (i.e., code modules) under various conditions, revealing important properties of the code's run-time behavior that cannot be predicted solely through static analysis

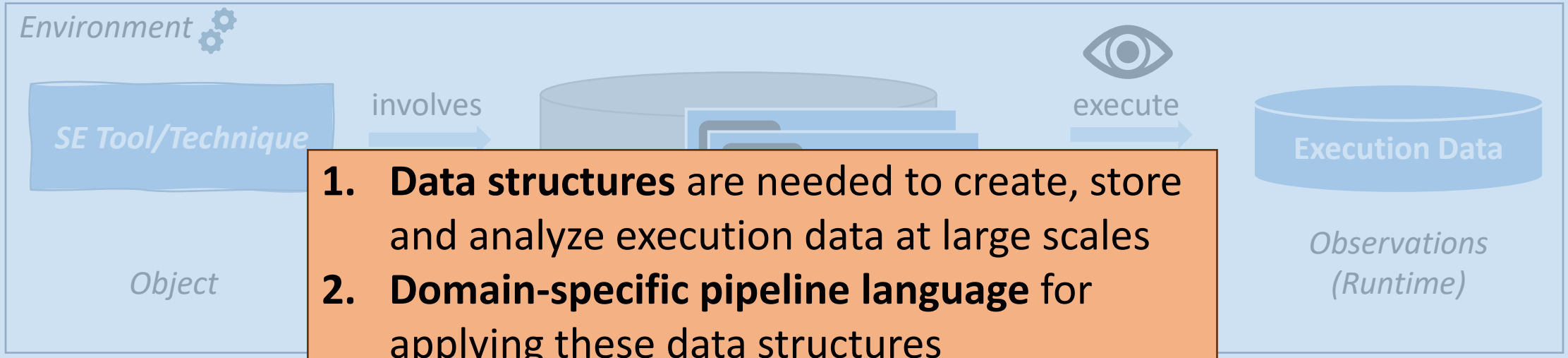
# Open Science Challenges in TDSEs




## Reproducibility of TDSEs is hindered by

- ad hoc scripting and custom code (general-purpose languages)
- ad hoc representation of tests and execution data (beyond pass/fail)
- lack of executable corpora
- complex execution logistics

# Open Science Challenges in TDSEs




1. **Data structures** are needed to create, store and analyze execution data at large scales
2. **Domain-specific pipeline language** for applying these data structures
3. **A dedicated platform** to execute these pipelines (automation) and store code/data

Reproducibilit

- ad hoc scripting and custom code (general-purpose languages)
- ad hoc representation of tests and execution data (beyond pass/fail)
- lack of executable corpora
- complex execution logistics

# Tabular Data Structures and DSL Languages for TDSEs

UNIVERSITÄT MANNHEIM

*Problem:* GCD (Greatest Common Divisor)
Interface: *gcd(int, int)->int*

`test(p1:Class, p2:int, p3:int)`

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | | create | ?p1 | | |
| 2 | | gcd | A1 | ?p2 | ?p3 |

*output* *operation* *inputs*

***Stimulus Sheet***

***Actuation Sheet***

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | | create | **V1** | | |
| 2 | 1 | gcd | A1 | 3 | 7 |

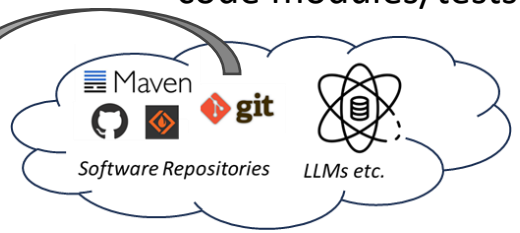
| | Time | Memory | Method Calls | Branches Covered | ... |
|---|---|---|---|---|---|
| | ... | ... | ... | ... | ... |

*Tests*

*N versions (candidate implementations)*

| | V1 | V2 | V3 | ... | $V_n$ |
|---|---|---|---|---|---|
| test1 | *test(V1,03,07)* | *test(V2, 03,07)* | *test(V3, 03,07)* | | *test($V_n$, 03,07)* |
| test2 | *test(V1,10,15)* | *test(V2,10,15)* | *test(V3,10,15)* | | *test($V_n$,10,15)* |
| test3 | *test(V1,49,14)* | *test(V2,49,14)* | *test(V3,49,14)* | | *test($V_n$,49,14)* |
| ... | | | | | |
| $test_m$ | *$test_m$(V1, x, y)* | *$test_m$(V2, x, y)* | *$test_m$(V3, x, y)* | | *$test_m$($V_n$, x, y)* |

code modules/tests




***Stimulus Matrix (SM)***

***Pipeline Script***

***DSL "LSL"***

***Test Driver "Arena"***

***Stimulus Response Matrix (SRM)***

| | V1 | | V2 | | V3 | | ... | $V_n$ | |
|---|---|---|---|---|---|---|---|---|---|
| test1 | *test(V1,03,07)* | *1* | *test(V2,03,07)* | *7* | *test(V3,03,07)* | *-7* | | *test($V_n$,03,07)* | ... |
| test2 | *test(V1,10,15)* | *5* | *test(V2,10,15)* | *15* | *test(V3,10,15)* | *-15* | | *test($V_n$,10,15)* | ... |
| test3 | *test(V1,49,14)* | *7* | *test(V2,49,14)* | *14* | *test(V3,49,14)* | *28* | | *test($V_n$,49,14)* | ... |
| ... | | | | | | | | | |
| $test_m$ | *$test_m$(V1, x, y)* | ... | *$test_m$(V2, x, y)* | ... | *$test_m$(V3, x, y)* | ... | | *$test_m$($V_n$, x, y)* | ... |

Differential Testing (identify discrepancies)

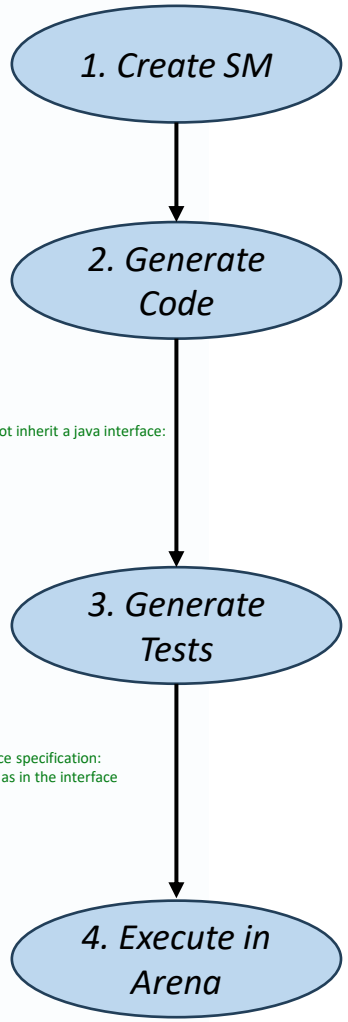
***Platform***

LASSO Platform

# Pipelines – LASSO Scripting Language (LSL)

UNIVERSITÄT MANNHEIM

```
dataSource 'lasso_quickstart'
study(name: 'OpenAI-DGAI') {

  profile('java17Profile') {
    scope('class') { type = 'class' }
    environment('java17') {
      image = 'maven:3.9-eclipse-temurin-17'
    }
  }

  def humanEval = loadBenchmark("humaneval-java-reworded")

  action(name: "createStimulusMatrices") {
    execute {
      def myProblems = [humanEval.abstractions['HumanEval_13_greatest_common_divisor']]
      myProblems.each { problem ->
        stimulusMatrix(problem.id, problem.lql, [/*impls*/], problem.tests) // id, interface, impls, tests
      }
    }
  }

  action(name: 'generateCodeGpt', type: 'GenerateCodeOpenAI') {
    dependsOn 'createStimulusMatrices'
    include '*'
    profile('java17Profile')

    apiKey = "demo"
    model = "gpt-4o-mini"
    samples = 1

    prompt { stimulusMatrix ->
      def prompt = [:] // create prompt model
      prompt.promptContent = """implement a java class with the following interface specification, but do not inherit a java interface:
```${stimulusMatrix.lql}```. Only output the java class and nothing else."""
      return [prompt] // list of prompts
    }
  }

  action(name: 'generateTestsGpt', type: 'GenerateTestsOpenAI') {
    dependsOn 'generateCodeGpt'
    include '*'
    profile('java17Profile')

    apiKey = "demo"
    model = "gpt-4o-mini"
    samples = 1

    prompt { stimulusMatrix ->
      def prompt = [:] // create prompt model
      prompt.promptContent = """generate a junit test class to test the functionality of the following interface specification:
```${stimulusMatrix.lql}```. Assume that the specification is encapsulated in a class that uses the same naming as in the interface
specification. Only output the JUnit test class and nothing else."""
      return [prompt] // list of prompts
    }
  }

  action(name: 'execute', type: 'Arena') {
    features = ["cc"] // coverage: code coverage, mutation testing etc.

    dependsOn 'generateTestsGpt'
    include '*'
    profile('java17Profile')
  }
}
```




| |
|---|
| test1 |
| test2 |
| test3 |

New **Stimulus Matrix (SM)** from coding problem

| | V1 | V2 | … | $V_n$ |
|---|---|---|---|---|
| test1 | | | | |
| test2 | | | | |
| test3 | | | | |

*Columns* – Add **implementations** (versions)

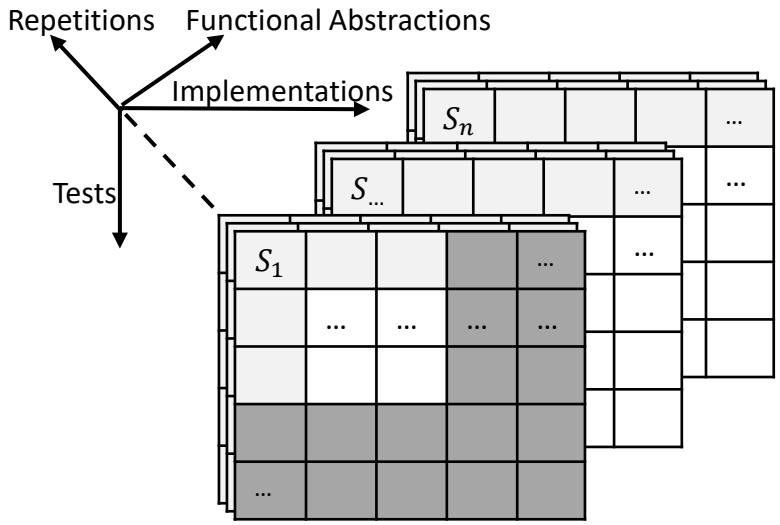
| | V1 | V2 | … | $V_n$ |
|---|---|---|---|---|
| test1 | | | | |
| test2 | | | | |
| … | | | | |
| $test_m$ | | | | |

*Rows* – Add **tests**

| | V1 | | V2 | | … | $V_n$ | |
|---|---|---|---|---|---|---|---|
| test1 | *test(V1,03,07)* | *1* | *test(V2,03,07)* | *7* | | *test($V_n$,03,07)* | … |
| test2 | *test(V1,10,15)* | *5* | *test(V2,10,15)* | *15* | | *test($V_n$,10,15)* | … |
| … | | | | | | | |
| $test_m$ | *$test_m$(V1, x, y)* | … | *$test_m$(V2, x, y)* | … | | *$test_m$($V_n$, x, y)* | … |

Observe/store **outputs**, **metrics** …

# Data-driven Analysis of SRMs




*Collections of SRMs*
*(navigation and dimensions)*

*Oracle values or reference implementation*

SM

| | Oracle | V1 |
|---|---|---|
| test1 | *test(V1,03,07)* | *test(V1,03,07)* |
| test2 | *test(V1,10,15)* | *test(V1,10,15)* |
| test3 | *test(V1,49,14)* | *test(V1,49,14)* |

execute

SRM

| | Oracle | | V1 | | ... |
|---|---|---|---|---|---|
| test1 | *test(V1,03,07)* | *1* | *test(V1,03,07)* | *1* | |
| test2 | *test(V1,10,15)* | *5* | *test(V1,10,15)* | *5* | |
| test3 | *test(V1,49,14)* | *7* | *test(V1,49,14)* | *11* | |

*Compare observed outputs (columns)*

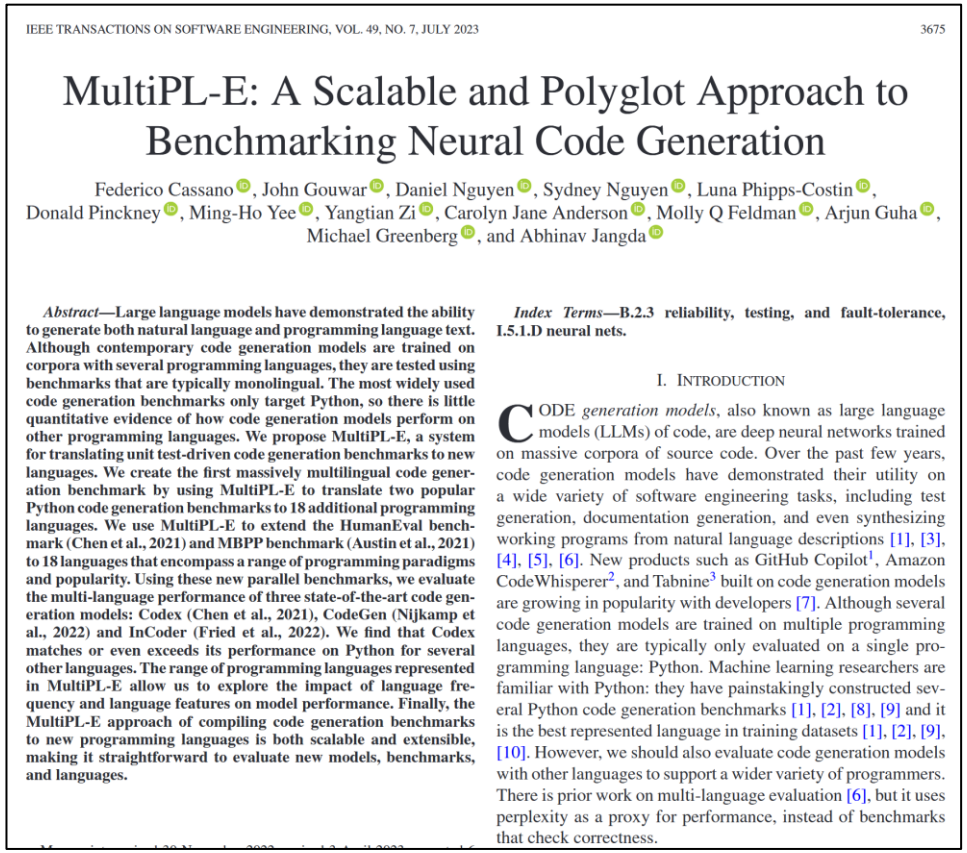
*Data-Driven Analysis*
*(incl. differential testing and*
*judging functional equivalence)*

# Demonstration – Reproducing TDSEs with LASSO

UNIVERSITÄT MANNHEIM

<table>
<tr><th rowspan="2">Purpose/ Design<br>Team</th><th colspan="2">Same Purpose (i.e., RQs)</th><th>Different Purpose (i.e., RQs)</th></tr>
<tr><th>Same Design</th><th>Different Design</th><th rowspan="3">Repurposing:<br>• different purpose (i.e., RQs)<br>• original or independent team<br>• different (logical) design<br>• performed in a different way</th></tr>
<tr><td>Original Team</td><td>Repetition:<br>• same purpose<br>• original team<br>• same (logical) design</td><td rowspan="2">Reproduction:<br>• same purpose<br>• original or independent team<br>• different (logical) design</td></tr>
<tr><td>Different Team</td><td>Replication:<br>• same purpose<br>• independent team<br>• same (logical) design</td></tr>
</table>

# TDSE Replication – Benchmarking LLMs for Code Generation



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 49, NO. 7, JULY 2023 3675

## MultiPL-E: A Scalable and Polyglot Approach to Benchmarking Neural Code Generation

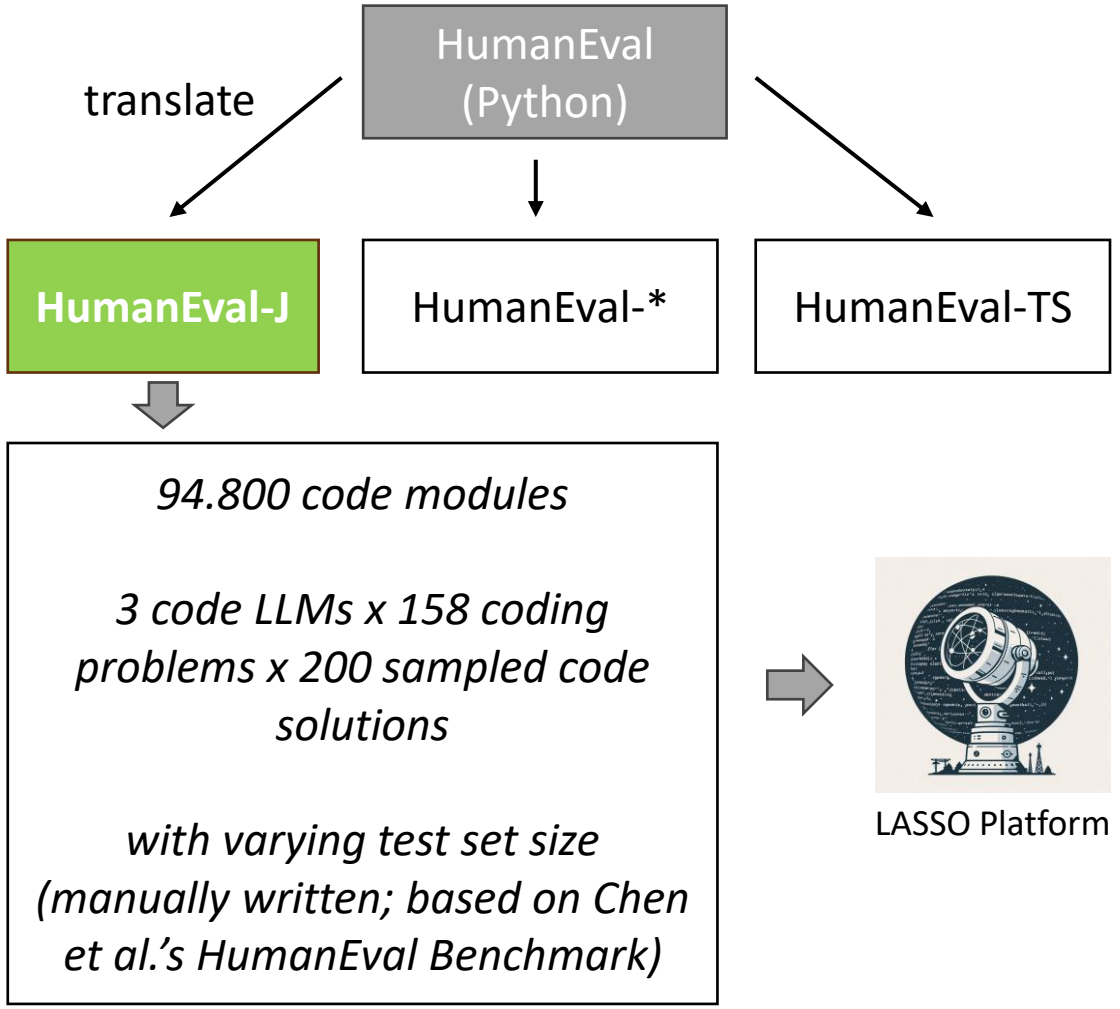
Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda

***Abstract*—Large language models have demonstrated the ability to generate both natural language and programming language text. Although contemporary code generation models are trained on corpora with several programming languages, they are tested using benchmarks that are typically monolingual. The most widely used code generation benchmarks only target Python, so there is little quantitative evidence of how code generation models perform on other programming languages. We propose MultiPL-E, a system for translating unit test-driven code generation benchmarks to new languages. We create the first massively multilingual code generation benchmark by using MultiPL-E to translate two popular Python code generation benchmarks to 18 additional programming languages. We use MultiPL-E to extend the HumanEval benchmark (Chen et al., 2021) and MBPP benchmark (Austin et al., 2021) to 18 languages that encompass a range of programming paradigms and popularity. Using these new parallel benchmarks, we evaluate the multi-language performance of three state-of-the-art code generation models: Codex (Chen et al., 2021), CodeGen (Nijkamp et al., 2022) and InCoder (Fried et al., 2022). We find that Codex matches or even exceeds its performance on Python for several other languages. The range of programming languages represented in MultiPL-E allow us to explore the impact of language frequency and language features on model performance. Finally, the MultiPL-E approach of compiling code generation benchmarks to new programming languages is both scalable and extensible, making it straightforward to evaluate new models, benchmarks, and languages.**

***Index Terms*—B.2.3 reliability, testing, and fault-tolerance, I.5.1.D neural nets.**

### I. Introduction

CODE *generation models*, also known as large language models (LLMs) of code, are deep neural networks trained on massive corpora of source code. Over the past few years, code generation models have demonstrated their utility on a wide variety of software engineering tasks, including test generation, documentation generation, and even synthesizing working programs from natural language descriptions [1], [3], [4], [5], [6]. New products such as GitHub Copilot[1], Amazon CodeWhisperer[2], and Tabnine[3] built on code generation models are growing in popularity with developers [7]. Although several code generation models are trained on multiple programming languages, they are typically only evaluated on a single programming language: Python. Machine learning researchers are familiar with Python: they have painstakingly constructed several Python code generation benchmarks [1], [2], [8], [9] and it is the best represented language in training datasets [1], [2], [9], [10]. However, we should also evaluate code generation models with other languages to support a wider variety of programmers. There is prior work on multi-language evaluation [6], but it uses perplexity as a proxy for performance, instead of benchmarks that check correctness.



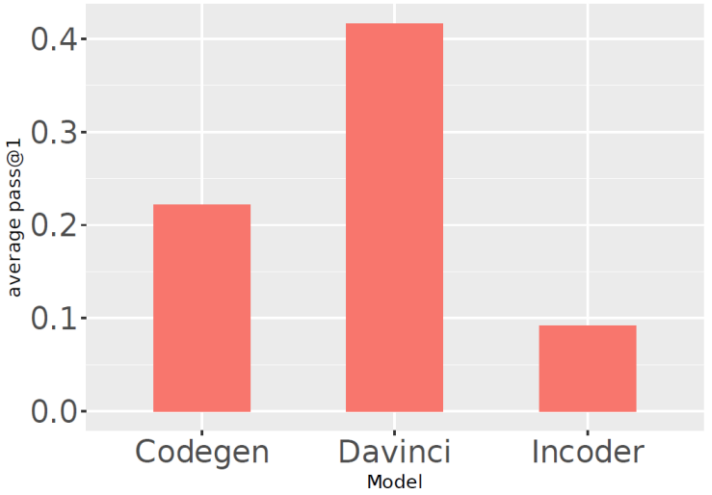
# Replication – Judging Functional Correctness

UNIVERSITÄT MANNHEIM

***RQ1**: What is the Java code completion performance of the three code models (with temperature set to 0.2) as judged by functional correctness? (pass@k metric)*

*Same design, same RQ*

| | $O$ | $S_{1_{LLM_1}}$ | ... | $S_{k_{LLM_1}}$ | ... | $S_{1_{LLM_n}}$ | ... | $S_{k_{LLM_n}}$ |
|---|---|---|---|---|---|---|---|---|
| $T_1$ | $T_1(0, \dots)$ | $T_1(S_{1_{LLM_1}}, \dots)$ | ... | $T_1(S_{k_{LLM_1}}, \dots)$ | ... | $T_1(S_{1_{LLM_n}}, \dots)$ | ... | $T_1(S_{k_{LLM_n}}, \dots)$ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| $T_j$ | $T_j(O, \dots)$ | $T_j(S_{1_{LLM_1}}, \dots)$ | ... | $T_j(S_{k_{LLM_1}}, \dots)$ | ... | $T_j(S_{1_{LLM_n}}, \dots)$ | ... | $T_j(S_{k_{LLM_n}}, \dots)$ |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| $T_m$ | $T_m(O, \dots)$ | $T_m(S_{1_{LLM_1}}, \dots)$ | ... | $T_m(S_{k_{LLM_1}}, \dots)$ | ... | $T_m(S_{1_{LLM_n}}, \dots)$ | ... | $T_m(S_{k_{LLM_n}}, \dots)$ |




| Test@Invocation | Oracle | $\text{CodeGen}_1$ | $\text{CodeGen}_2$ | $\text{CodeGen}_3$ | $\text{CodeGen}_4$ |
|---|---|---|---|---|---|
| $T_1$@1 | 1 | 1 | −7 | 1 | 1 |
| $T_2$@1 | 5 | 5 | −15 | 5 | 5 |
| $T_3$@1 | 7 | 7 | 28 | 7 | 7 |
| $T_4$@1 | 12 | 12 | 60 | 12 | 12 |

*Judging functional correctness using SRMs*

# Replication – Judging Functional Correctness

RQ: What is the Java code completion performance of the three code models (with temperature set to 0.2) as judged by functional correctness? (pass@k metric)

| | $O$ | $S_{1_{LL}}$ | | $S$ | | $S$ | | $S$ |
|---|---|---|---|---|---|---|---|---|
| $T_1$ | $T_1(O, \dots)$ | $T_1(S_{1_{LL}}$ | | | | | | |
| … | … | … | | | | | | |
| $T_j$ | $T_j(O, \dots)$ | $T_j(S_{1_{LL}}$ | | | | | | |
| … | … | … | | | | | | |
| $T_m$ | $T_m(O, \dots)$ | $T_m(S_{1_{LLM_1}}, \dots)$ | … | $T_m(S_{k_{LLM_1}}, \dots)$ | … | $T_m(S_{1_{LLM_n}}, \dots)$ | … | $T_m(S_{k_{LLM_n}}, \dots)$ |

**Successful replication of original TDSE**

→ Replication resulted in one LSL study pipeline script (LOC < 50)





| Test@Invocation | Oracle | $\text{CodeGen}_1$ | $\text{CodeGen}_2$ | $\text{CodeGen}_3$ | $\text{CodeGen}_4$ |
|---|---|---|---|---|---|
| $T_1$@1 | 1 | 1 | −7 | 1 | 1 |
| $T_2$@1 | 5 | 5 | −15 | 5 | 5 |
| $T_3$@1 | 7 | 7 | 28 | 7 | 7 |
| $T_4$@1 | 12 | 12 | 60 | 12 | 12 |

*Judging functional correctness using SRMs*

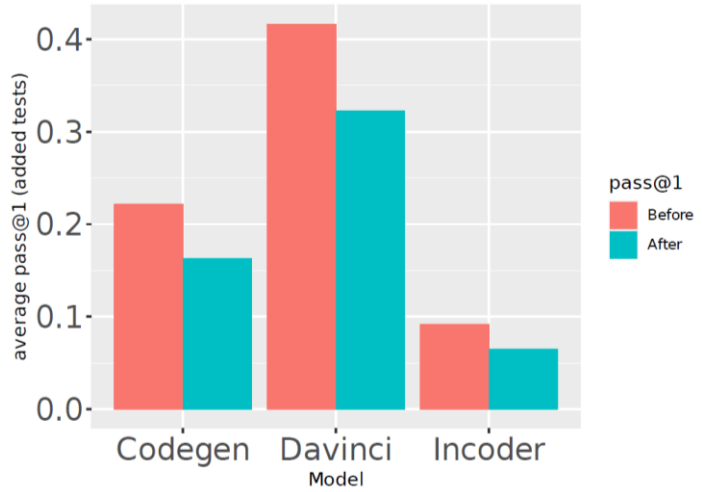
# Reproduction – Adding more Tests (EvoSuite)

UNIVERSITÄT MANNHEIM

***RQ1**: What is the Java code completion performance of the three code models (with temperature set to 0.2) as judged by functional correctness ? (pass@k metric)*

*Different design, same RQ*

| | $S_{1_{LLM_1}}$ | … | $S_{k_{LLM_1}}$ | … | $S_{1_{LLM_n}}$ | … | $S_{k_{LLM_n}}$ |
|---|---|---|---|---|---|---|---|
| $T_1$ | $T_1(S_{1_{LLM_1}}, \dots)$ | … | $T_1(S_{k_{LLM_1}}, \dots)$ | … | $T_1(S_{1_{LLM_n}}, \dots)$ | … | $T_1(S_{k_{LLM_n}}, \dots)$ |
| … | … | … | … | … | … | … | … |
| $T_j$ | $T_j(S_{1_{LLM_1}}, \dots)$ | … | $T_j(S_{k_{LLM_1}}, \dots)$ | … | $T_j(S_{1_{LLM_n}}, \dots)$ | … | $T_j(S_{k_{LLM_n}}, \dots)$ |
| … | … | … | … | … | … | … | … |
| $T_m$ | $T_m(S_{1_{LLM_1}}, \dots)$ | … | $T_m(S_{k_{LLM_1}}, \dots)$ | … | $T_m(S_{1_{LLM_n}}, \dots)$ | … | $T_m(S_{k_{LLM_n}}, \dots)$ |
| $E_1$ | $E_1(S_{1_{LLM_1}}, \dots)$ | … | $E_1(S_{k_{LLM_1}}, \dots)$ | … | $E_1(S_{1_{LLM_n}}, \dots)$ | … | $E_1(S_{k_{LLM_n}}, \dots)$ |
| … | … | … | … | … | … | … | … |
| $E_n$ | $E_n(S_{1_{LLM_1}}, \dots)$ | … | $E_n(S_{k_{LLM_1}}, \dots)$ | … | $E_n(S_{1_{LLM_n}}, \dots)$ | … | $E_n(S_{k_{LLM_n}}, \dots)$ |




Adding more **tests** to SRMs using *EvoSuite* (Fraser and Arcuri)

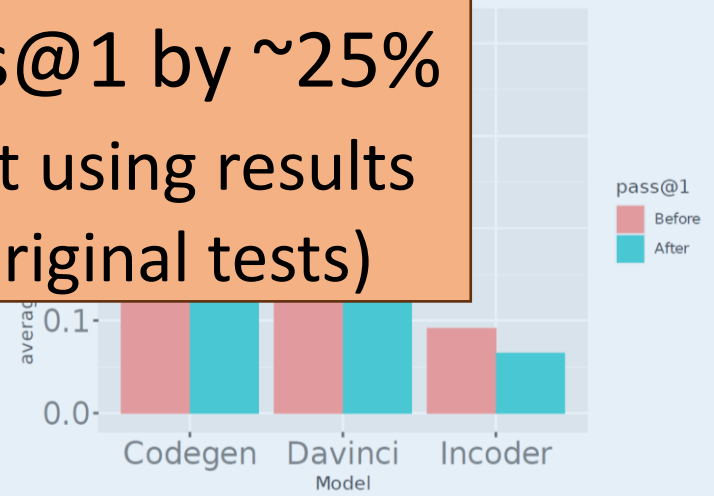
# Reproduction – Adding more Tests (EvoSuite)

RQ: What is the Java code completion performance of the three code models (with temperature set to 0.2) as judged by functional correctness ? (pass@k metric)

Aligns with findings of related TDSEs (e.g., *EvalPlus*; Liu et al.) – decrease of pass@1 by ~25%

→ Reproduction resulted in one LSL script using results from previous replication (no re-run of original tests)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $T_1$ | $T_1($ | | | | | | |
| ... | ... | | | | | | |
| $T_j$ | $T_j($ | | | | | | |
| ... | ... | | | | | | |
| $T_m$ | $T_m$ | | | | | | |
| $E_1$ | $E_1(S_{1_{LLM_1}}, \dots)$ | ... | $E_1(S_{k_{LLM_1}}, \dots)$ | ... | $E_1(S_{1_{LLM_n}}, \dots)$ | ... | $E_1(S_{k_{LLM_n}}, \dots)$ |
| ... | ... | ... | ... | ... | ... | ... | ... |
| $E_n$ | $E_n(S_{1_{LLM_1}}, \dots)$ | ... | $E_n(S_{k_{LLM_1}}, \dots)$ | ... | $E_n(S_{1_{LLM_n}}, \dots)$ | ... | $E_n(S_{k_{LLM_n}}, \dots)$ |




Adding more **tests** to SRMs using EvoSuite (Fraser and Arcuri)

# Repurposing – Behavioral Clustering

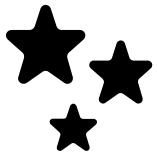
UNIVERSITÄT MANNHEIM

***RQ2**: What is the variability in observable functional behavior exhibited by the software components sampled from the code models?*

*Same design, different RQ*

| | $S_{1_{LLM_1}}$ | ... | $S_{k_{LLM_1}}$ | ... | $S_{1_{LLM_n}}$ | ... | $S_{k_{LLM_n}}$ |
|---|---|---|---|---|---|---|---|
| $T_1$ | $T_1(S_{1_{LLM_1}}, \dots)$ | ... | $T_1(S_{k_{LLM_1}}, \dots)$ | ... | $T_1(S_{1_{LLM_n}}, \dots)$ | ... | $T_1(S_{k_{LLM_n}}, \dots)$ |
| ... | ... | ... | ... | ... | ... | ... | ... |
| $T_j$ | $T_j(S_{1_{LLM_1}}, \dots)$ | ... | $T_j(S_{k_{LLM_1}}, \dots)$ | ... | $T_j(S_{1_{LLM_n}}, \dots)$ | ... | $T_j(S_{k_{LLM_n}}, \dots)$ |
| ... | ... | ... | ... | ... | ... | ... | ... |
| $T_m$ | $T_m(S_{1_{LLM_1}}, \dots)$ | ... | $T_m(S_{k_{LLM_1}}, \dots)$ | ... | $T_m(S_{1_{LLM_n}}, \dots)$ | ... | $T_m(S_{k_{LLM_n}}, \dots)$ |

Cluster implementations (SRM columns)
by the equivalence of their observed outputs



Descriptive statistics of cluster sizes over all functional abstractions (research question RQ 2).

| | min | q25 | median | q75 | max | mean | sd |
|---|---|---|---|---|---|---|---|
| Total | 1 | 11.00 | 23.50 | 49.75 | 143 | 32.71 | 29.25 |
| Codegen | 1 | 3.00 | 9.00 | 21.00 | 80 | 14.45 | 16.04 |
| Davinci | 1 | 2.00 | 4.00 | 7.00 | 55 | 5.83 | 7.39 |
| InCoder | 1 | 6.00 | 12.00 | 24.00 | 76 | 17.61 | 16.05 |

# Repurposing – Behavioral Clustering

***RQ2**: What is the variability in observable functional behavior exhibited by the software components sampled from the code models?*

| | $S_{1_{LLM_1}}$ | … |
|---|---|---|
| $T_1$ | $T_1(S_{1_{LLM_1}}, \dots)$ | … |
| … | … | … |
| $T_j$ | $T_j(S_{1_{LLM_1}}, \dots)$ | … |
| … | … | … |
| $T_m$ | $T_m(S_{1_{LLM_1}}, \dots)$ | … |

*Davinci* provides the least variability in functional behavior (~5.8 clusters on average)

→ No additional LSL script necessary, since SRMs of previous study were analyzed directly

Cluster implementations (SRM columns)
by the equivalence of their observed outputs

ractions (research question

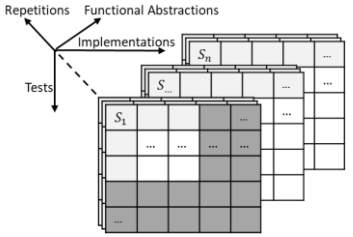
| x | mean | sd |
|---|---|---|
| | 32.71 | 29.25 |
| | 14.45 | 16.04 |
| | 5.83 | 7.39 |
| | 17.61 | 16.05 |

# Main Takeaways

- We successfully replicated, reproduced and repurposed a popular TDSE (i.e., benchmarking LLMs for code generation – HumanEval-J)
- Resulting LSL study pipeline scripts
  - encode study designs and assumptions
  - can be shared with others together with SRMs
  - can be repeated by executing it again

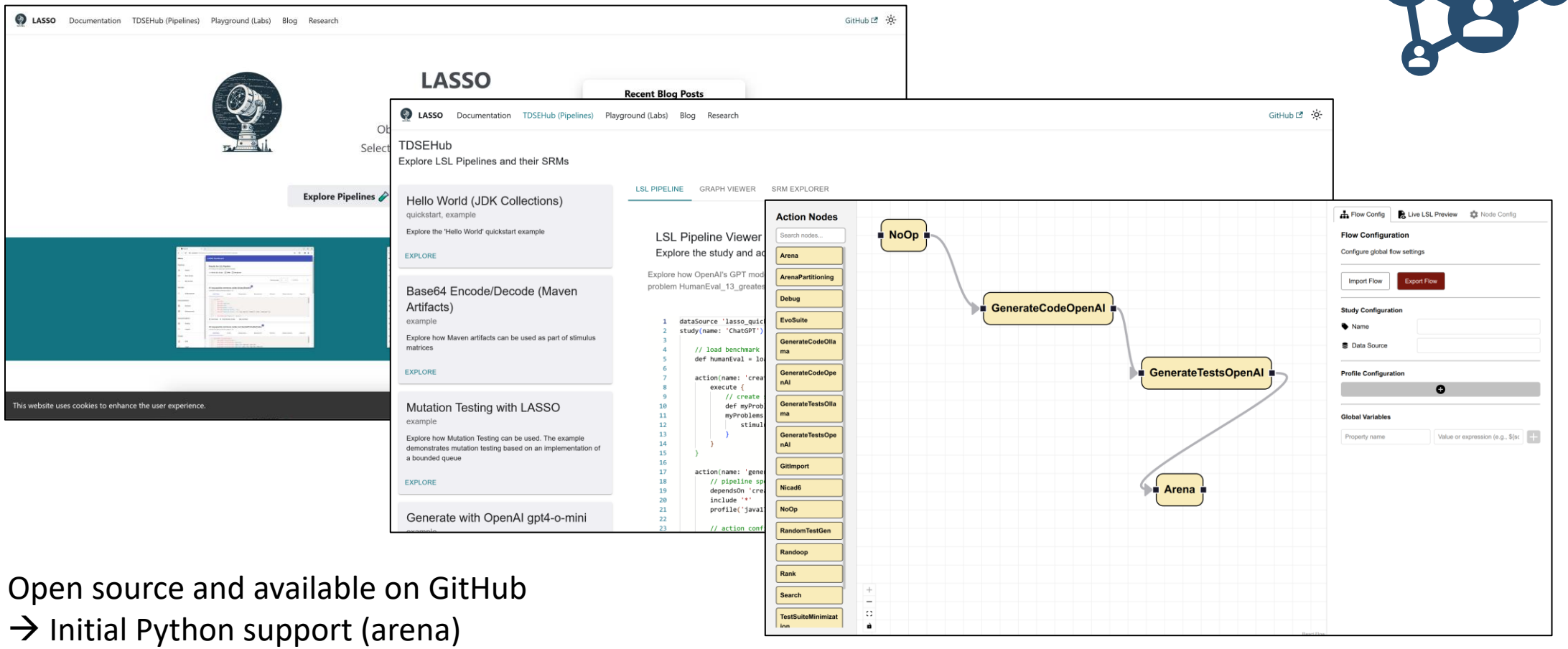
  → Increases transparency + fosters open science principles
- Limitations
  - serialization of objects (currently: JSON string representation)
  - equivalence checking in tabular representations

*LSL Script*



# LASSO Community

- https://softwareobservatorium.github.io




Open source and available on GitHub
→ Initial Python support (arena)

# Thank you!

Feel free to reach out ☺

The Journal of Systems and Software 212 (2024) 111971

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Promoting open science in test-driven software experiments☆

Marcus Kessel *, Colin Atkinson

University of Mannheim, 68159 Mannheim, Germany

ARTICLE INFO

*Keywords:*
Software
Engineering
Empirical
Experimentation
Observation
Behavior
Reproducibility
Replication
Data structures
Open science
Large language models
Machine learning
Generative artificial intelligence
Benchmark
Language-to-code
HumanEval
Automation
Measurement

ABSTRACT

A core principle of open science is the clear, concise and accessible publication of empirical data, including "raw" observational data as well as processed results. However, in empirical software engineering there are no established standards (de jure or de facto) for representing and "opening" observations collected in test-driven software experiments — that is, experiments involving the execution of software subjects in controlled scenarios. Execution data is therefore usually represented in ad hoc ways, often making it abstruse and difficult to access without significant manual effort. In this paper we present new data structures designed to address this
soft
pro
too
set
ing the stimuli and responses used to execute
their utility, we show how they can be used to
imental evaluations of AI-based code completion
ate the incremental expansion of execution data
s addressing new research questions.

1. Introduction

Empirical studies have grown in importance
ing research over the last few years as leading
conferences and journals have placed increased
for research claims to be supported by empiric
et al., 2015; ESE, 2023). Most recently, this trend has evolved to encompass the principles of "Open Science" which aims to facilitate transparency, collaboration, and accessibility in software engineering research by making research data, methods and findings publicly available (Méndez Fernández et al., 2019). Today, open science is actively promoted by most leading software engineering conferences and journals such as EMSE[1], and has spawned new research communities such as the "Empirical Software Engineering and Measurement" (ESEM) symposium (ESEM, 2023).

The basic motivation for this trend is broadly the same across all scientific and engineering disciplines — namely, making all elements of empirical studies available for other researchers to examine, use and build upon, so that research communities can reach consensus

more easily repeat, replicate and reproduce
ACM (ACM, 2023), repetition involves the
g an empirical study with the original setup,
erent team reperforming an empirical study
nd reproduction involves a different team
study with a different design and/or setup.

The fundamental principles and ingredients of open science are also basically the same across scientific disciplines, even though the terminology and the detailed breakdown of steps may differ. In the social science domain, Minocher et al. (2020) define four fundamental prerequisites for an empirical study to be fully reproducible – "(1) data recoverability, the availability of the involved data and other materials to attempt reproduction of analyses, (2) data usability, the completeness and clarity of data, when available, (3) analytical clarity, the adequacy of published reports for repeating analyses, and (4) agreement of reproduced results with published results". In their overview of open science in software engineering, Mendez et al. (2020) point out that the first of these, data recoverability, involves the dissemination

☆ Editor: Alexander Serebrenik.
* Corresponding author.
*E-mail addresses:* marcus.kessel@uni-mannheim.de (M. Kessel), colin.atkinson@uni-mannheim.de (C. Atkinson).
[1] with its Open Science Initiative (https://emsejournal.github.io/open_science)

https://doi.org/10.1016/j.jss.2024.111971
Received 2 August 2023; Received in revised form 8 December 2023; Accepted 15 January 2024
Available online 12 March 2024
0164-1212/