

Liquid Types for AI

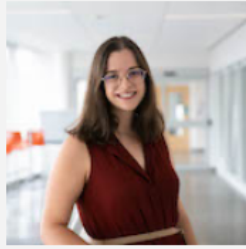
Alcides Fonseca

<http://alcidesfonseca.com/>

LASIGE & Faculdade de Ciências da Universidade de Lisboa

Alcides Fonseca et al.

PhD Students



Catarina Gamboa — Improving the Usability of Liquid Types for Reliable Software (with Jonathan Aldrich)

- ☐ Latte: Lightweight Aliasing Tracking for Java at HATRA+IWACO 2023
- ☐ Usability-Oriented Design of Liquid Types for Java at ICSE 2023
- 🏛️ Student Volunteer Chair at SPLASH 2023 and SPLASH 2024
- 🏛️ Artifact Evaluation Committee at PLDI 2023



Paulo Santos — Specification-Guided Repair for Robotic Systems (with Chris Timperley and Sara Silva)

- ☐ Is it a Bug? Understanding Physical Unit Mismatches in Robot Software at ICRA 2024
- ☐ An Experience Report on Challenges in Learning the Robot Operating System at RoSE 2022
- 🔧 Where is the Bug? Understanding, detecting and repairing misconfigurations in ROS at RosCon 2023
- 🏛️ Web and Social Media Chair at FormalISE 2024



Guilherme Espada — Stochastic Resource Analysis of Programs

- 🏛️ Artifact Evaluation Committee at PLDI 2022
- 🏛️ Video Chair at PLDI 2022, POPL 2023, PLDI 2023, ICFP 2023, SPLASH 2023, POPL 2024
- 🏢 Internships at Feedzai, Decipad



Pedro Barbosa — Modelling the impact of deep intronic mutations on the human disease (with Carmo-Fonseca)

- ☐ Computational prediction of human deep intronic variation at GigaScience
- ☐ Clinical significance of genetic variation in hypertrophic cardiomyopathy: comparison of computational tools to prioritize missense variants at Frontiers in Cardiovascular Medicine [its](#)

Interns



Leon Ingelse — GeneticEngine, an hybrid grammar-guided and strongly-typed genetic programming framework

- ☐ Comparing Individual Representations in Grammar-Guided Genetic Programming for Glucose Prediction in People with Diabetes at GEWS @ Gecco 2023
- ☐ Domain-Aware Feature Learning with Grammar-Guided Genetic Programming at EuroGP 2023

Foundations of TypeSystems

Traditional Types

grade: **int** = 18

grade = 4.5

grade = -5

grade = 23

Refined Types

`grade: {x: Int | 0 <= x && x <= 20} = 18`

`grade = 4.5`

`grade = -5`

`grade = 23`

$\{x:T|p\}$ is a **refined type**.

- x is the name of variable (or fresh name)
- T is a type (usually a base type)
- p is a boolean predicate that can mention x .

$\{x:\text{Int} \mid x > 0\}$
 $\{y:\text{Bool} \mid y\}$
 $\{z:\text{Float} \mid \text{true}\}$

Refined Types

```
def bonus (ograde:Intl0<=ograde<=20)
          : (ngrade:Intl0<=ngrade<=20){
    ograde + 1
}
```

```
def bonus (ograde:Intl0<=ograde<=20)
          : (ngrade:Intl0<=ngrade<=20){
    if ograde == 20 then ograde
    else ograde + 1
}
```

Dependent Types

```
def bonus (ograde:Int!0<=ograde<=20)
  : (ngrade:Int!0<=ngrade<=20
    && ngrade >= ograde){
  if ograde == 20 then ograde
    else ograde + 1
}
```

$\{x:T \mid p\}$ is a **refined and dependent** type.

- if p mentions variables other than x .

$$\{x:\text{Int} \mid x > y\} \\ \{p:\text{Int} \mid \text{true}\} \rightarrow \{q:\text{Int} \mid q > p\}$$

def div: (x:Int) (y:Intly!=0) : Int {...}

div 3 0 **X**

Subtyping:

$\{z:\text{Int} \mid z==0\} <: \{y:\text{intly} \neq 0\}$

Verification Condition:

$\forall v:\text{Int}, v==0 \rightarrow v \neq 0$

def div: (x:Int) (y:Intly!=0) : Int {...}

div 3 2 ✓

Subtyping:

$\{z:\text{Int} \mid z==2\} <: \{y:\text{intly} \neq 0\}$

Verification Condition:

$\forall v:\text{Int}, v==2 \rightarrow v \neq 0$

How to prove (or disprove)

$\forall v:\text{Int}, v \geq 0 \rightarrow v \neq 0$

?

Liquid Types: Discharging to an SMT Solver

Full Dependent Types: Manually

Foundations of Liquid Types

$\{x:T|p\}$ is a liquid, refined and dependent type.

- if p is drawn from a decidable logic (QF_UFLIA)

QF: Quantifier-Free

UF: Uninterpreted Functions

LIA: Linear Integer Arithmetic

$\{x:\text{Tip}\}$ is a **liquid, refined and dependent** type.

$\{z:\text{Int} \mid z > 0\}$
 $\{z:\text{Int} \mid z + y > 0\}$
 $\{z:\text{Int} \mid z * y > 0\}$
 $\{z:\text{Int} \mid z * 3 > 0\}$

Verification Condition:

$\forall v:\text{Int}, v==0 \rightarrow v!=0$

```
>>> import z3
>>> s = z3.Solver()
>>> v = z3.Int("v")
>>> s.add(v==2)
>>> s.add(z3.Not(v!=0))
>>> s.check()
unsat
>>> █
```

```
>>> import z3
>>> s = z3.Solver()
>>> v = z3.Int("v")
>>> s.add(v==0)
>>> s.add(z3.Not(v!=0))
>>> s.check()
sat
>>> s.model()
[v = 0]
>>> █
```

$\{x:\text{Tip}\}$ is a liquid, refined and dependent type.

Uninterpreted Function

predicate **size** : Array -> Int

empty : $\{z:\text{Array} \mid \text{size}(z)=0\}$

append : x:Array ->

el: Int ->

$\{y:\text{Array} \mid \text{size}(y)=\text{size}(x)+1\}$

Languages with Liquid Types

ML(1991)

C (2012)

Haskell (2014)

Scala (2016)

Java (2018)

Javascript (2012)

Typescript (2016)

Rust (2022)

LiquidJava



Compiler for Java annotated with Liquid Types

- <https://catarinagamboa.github.io/liquidjava.html>

```
@Refinement("positive > 0")
int positive;
positive = 50; //Correct
positive = -1; //Error
```

```
@RefinementAlias("RGB(int x) {x >= 0 && x <= 255}")
class MyColor{
    @Refinement("RGB(r)") int r;
    @Refinement("RGB(g)") int g;
    @Refinement("RGB(b)") int b;
}
```

```
@Refinement("_ >= 0 && _ <= n")
public static int absolute(int n) {
    if(0 <= n)
        return n;
    else
        return 0 - n;
}
```

```
@RefinementAlias("Percentage(int x) { x >= 0 && x <= 100}")
public class Grade{

    @Refinement("Percentage(_)")
    public static int addBonus(@Refinement("Percentage(_)")
                               int grade,
                               @Refinement("Percentage(_) && bonus < grade")
                               int bonus){

        if((bonus + grade) > 100)
            return 100;
        else
            return grade + bonus;
    }
}
```

LiquidJava



Liquid Types and Typestate

Typestate

Overview Package **Class** Use Tree Deprecated Index Help

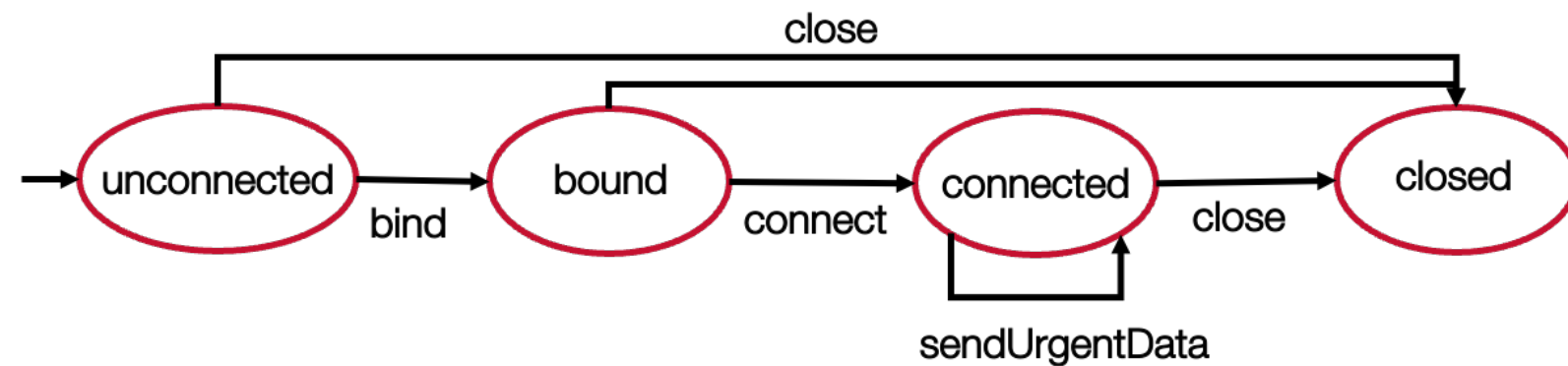
Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.net

Class Socket

java.lang.Object
java.net.Socket



```
@ExternalRefinementsFor ("java.net.Socket")
@StateSet ({ "unconnected", "bound", "connected", "closed" })
public interface SocketRefinements {

    @StateRefinement (to="unconnected(this)")
    public void Socket();

    @StateRefinement (from="unconnected(this)", to="bound(this)")
    public void bind (SocketAddress add);
```

Client code

```
Socket socket = new Socket();
socket.bind(new InetSocketAddress(...));
socket.sendUrgentData(90);
socket.close();
```

Programming Language with builtin Liquid Types

- <https://github.com/alcides/aeon>

```
type Unit;  
def math : Unit = native_import "math";  
def pow : (b:Int | 1<=b && b <= 100) ->  
    (d:Int | 1<=d && d <= 100) -> Int = native "lambda x: lambda y: math.pow(x , y)";  
  
def test (x:Int | x > 0) : (z:Int | z < 0) {  
    ?r  
}
```

Applications of Liquid Types

Machine Learning

Data Source

Data Transformation

PCA

Feature Engineering

Temporal Aggregation

Label Encoding

Label Decoding

One Hot Encoding

One Hot Decoding

Vertical Concatenation

Horizontal Concatenation

Data Balancing

Visualization

Classification

Regression

Classifier Evaluation

Regressor Evaluation

Import from CSV 1

creditcard.csv

Rows: 284807

Columns: 31

Time Series: false

Undersampling

Random State: None

Split Dataset

Test Size: 0.25

Train Size: 0.75

Shuffle: false

Stratify by Class: false

Random Forest Classifier

Trees: 100

Max Depth: None

Criterion: gini

Evaluate Classifier

Import from CSV 1

(Dataset Port n_cols == 31)

(Dataset Port n_rows == 284807)

(Dataset Port n_rows == 284807)

(Dataset Port time_series == False)

(Dataset Port reduced == False)

(Dataset Port processed == False)

Problems

Assertions

Log

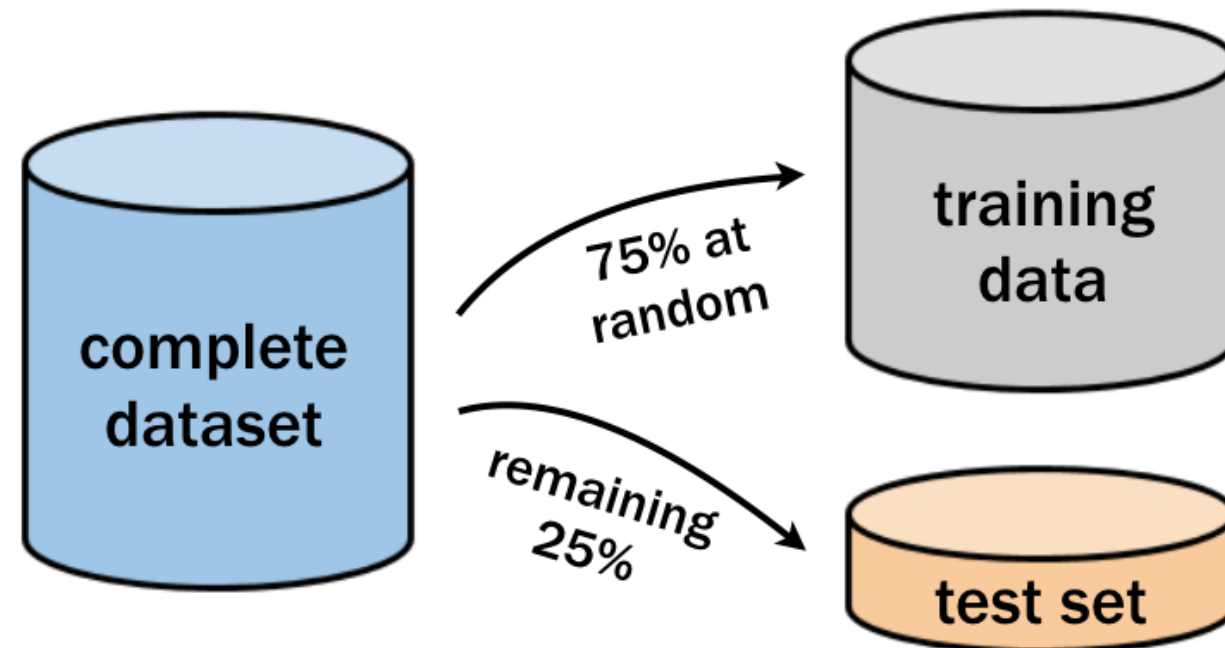
MLVP

Story #1: PHP Bytecode encoding

```
static const char *zend_vm_opcodes_names[199] = {  
    "ZEND_NOP",  
    "ZEND_ADD",  
    "ZEND_SUB",  
    "ZEND_MUL",  
    "ZEND_DIV",  
    "ZEND_MOD",  
  
    /* lots more */  
}
```

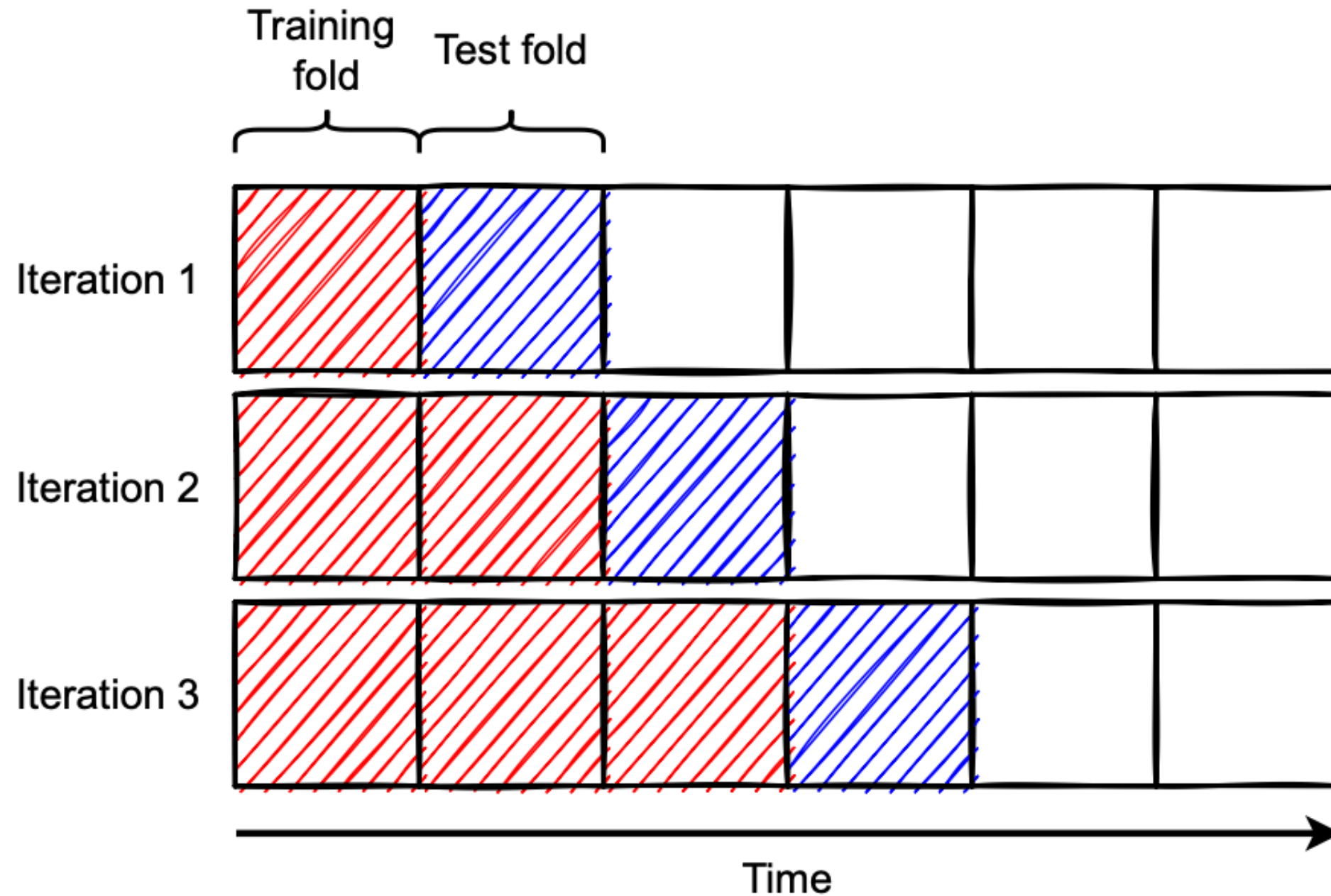
MLVP

Story #2: Sorted Dataset



MLVP

Story #3: Temporal Cross-Validation



MLVP

Story #4: Data Leakage

```
df['target'] = df['old_target'].map(  
    lambda x:1 if x == "Cat" else 0)  
X = df.drop('target')  
y = df['target']
```

⁵During our contamination check we discovered that portions of BIG-bench [48] were inadvertently mixed into the training set, and we excluded it from our reported results.

MLVP

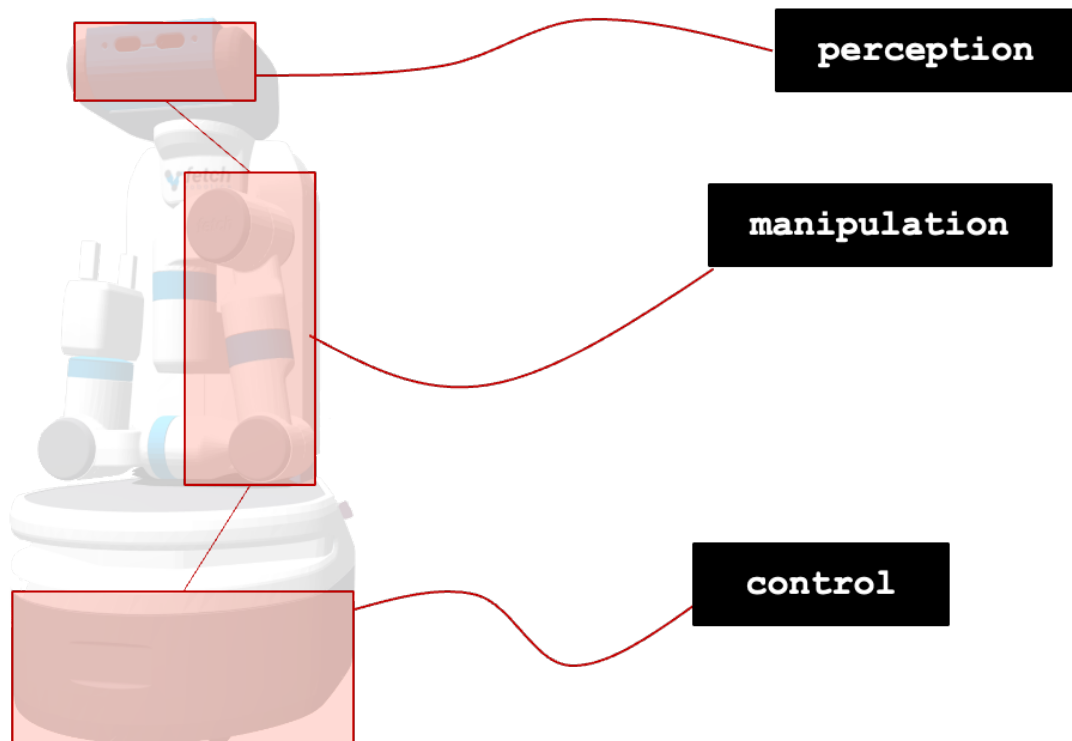
Static Verification of Semantic Properties

Categories	Defects
Feature normalization	Missing values
	Missing feature scaling
	Incorrect imputation
	Incorrect interpolation
Data balancing	Missing subsampling
	Missing supersampling
	Too many synthesized instances (e.g., SMOTE)
Training/testing instances	Use the same instances in training and testing
	Using features built from the target features in training
	Missing data shuffling
	Perform cross-validation using Time Series
Pipeline construction	Incompatible data type between components
	Unexpected data distribution
	Input dataset having an expected type or format

Applications of Liquid Types

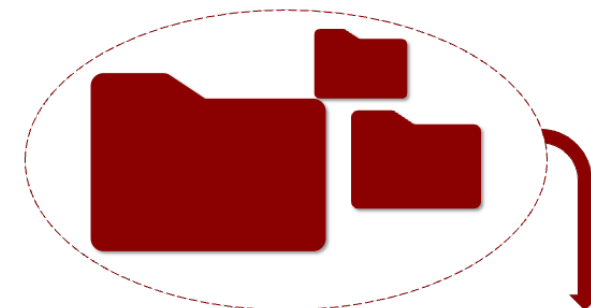
Robotics

Integrating Robotics Software Components is not easy!



High reuse of source code, libraries, and configuration files

[Garcia et al, 2020]



Developers rely on a small set of packages

[Estefo, 2019 et al; Kolak et al, 2020]

80% of dependencies

[Kolak et al, 2020]

Integrating Robotics Software Components is not easy!

Issues during development [Estefo et al, 2019; Afzal et al, 2020]

Difficulty in system configuration

Dependency Issues

Lack of documentation



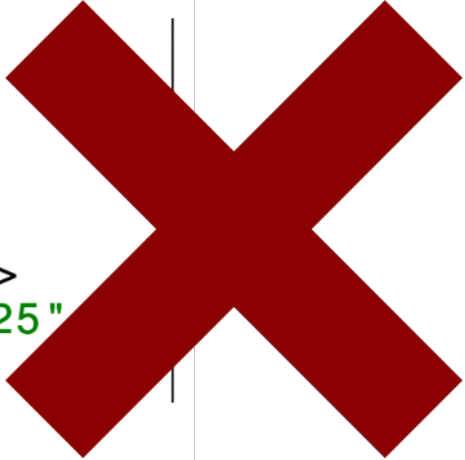
71%

reported issues when integrating new components

[Estefo et, 2019]

Integrating Robotics Software Components is not easy!

```
<!-- Launch joy node -->
<node pkg="joy" type="joy_node" name="joy">
  <param name="dev" value="$(arg_dev)" />
  <param name="deadzone" value="2" />
  <param name="autorepeat_rate" value="40" />
  <param name="coalesce_interval" value="0.025" />
</node>
```



~deadzone (double, default: 0.05)

Amount by which the joystick has to move before it is considered to be off-center. This parameter is specified relative to an axis normalized between -1 and 1. Thus, 0.1 means that the joystick has to move 10% of the way to the edge of an axis's range before that axis will output a non-zero value. Linux does its own deadzone processing, so in many cases this value can be set to zero.

ROSpec: a language for verification of robotics software architecture

Specification

```
expects {  
    cmdvel.linear.x > 0.0,  
    cmdvel.linear.y > 0.0,  
    cmdvel.linear.z > 0.0,  
} where {  
    cmdvel: Twist := topic /cmd_vel  
}
```

```
ensures {  
    topic_scan.range_max - topic_scan.range_min < ranges_size,  
    topic_scan.angle_min < topic_scan.angle_max,  
    topic_scan.range_min < topic_scan.range_max,  
    topic_scan.angle_increment >= 0,  
} where {  
    ranges_size: int := 120,  
    topic_scan: nav_msgs/LaserScan := topic /base_scan  
}
```

Native Functions

```
ensures {  
    tag(topic_scan, "metric"),  
    tag(t1, "metric"),  
} where {  
    t1: PoseWithCovariance := topic /initialpose  
}
```

LiquidJava for MAVSDK



- MAVSDK is one of the two most popular open-source drone controller frameworks (with a version in Java)



```
@ExternalRefinementsFor("io.mavsdk.mission.Mission")
@StateSet({...,"connectionEstablished", "missionUploaded", "missionStarted"})
public interface MissionControllerRefinements {
    @StateRefinement(from="connectionEstablished(this)",to="missionUploaded(this)")
    public void uploadMission(MissionPlan missionPlan);

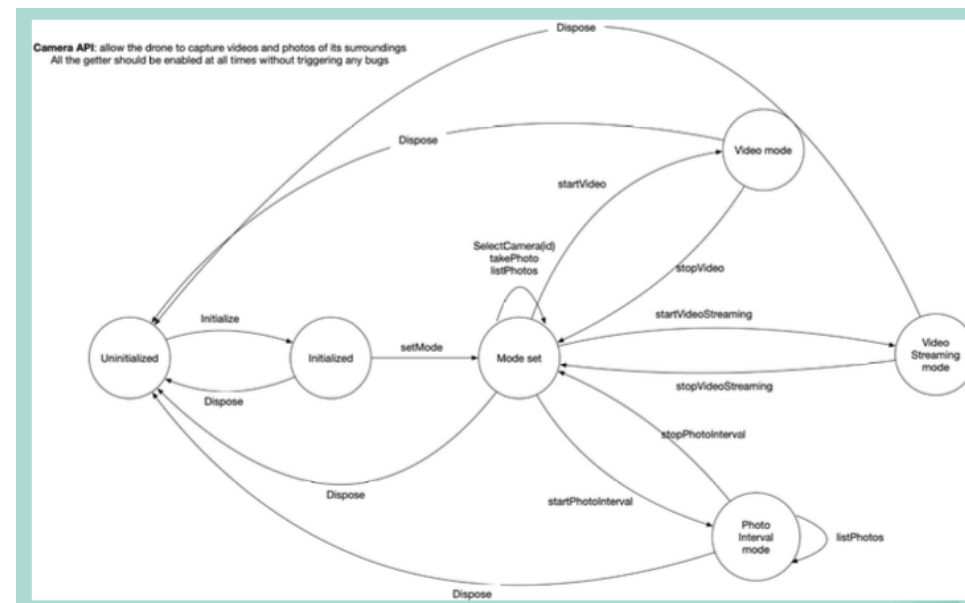
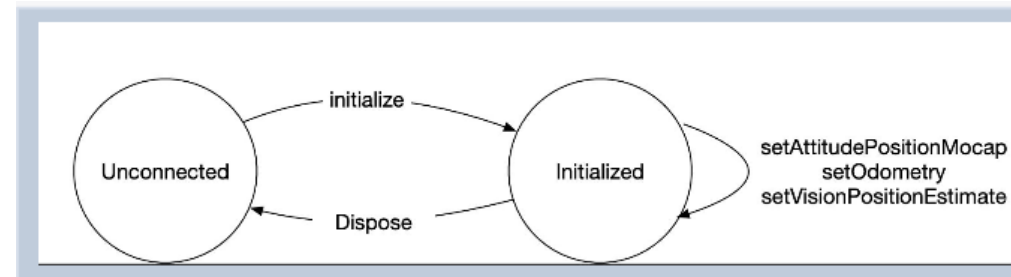
    @StateRefinement(from="missionUploaded(this)", to="missionStarted(this)")
    public void startMission();

    ...
}
```

```
Mission mission =
    drone.getMission();
//mission.uploadMission(...);
✗ mission.startMission();
```

LiquidJava for MAVSDK

	States	Transitions
Mocap	2	6
Geofence	3	5
Mission	4	14
Offboard	4	17
Camera	6	15
FTP	5	19
FollowMe	6	20



4 bugs were found!

2 in **Camera**, 1 in **FTP** and 1 in **FollowMe**

Why are
Liquid Types
not being adopted?

Participative Design of LiquidJava

Syntax Survey with 50 collected answers

A

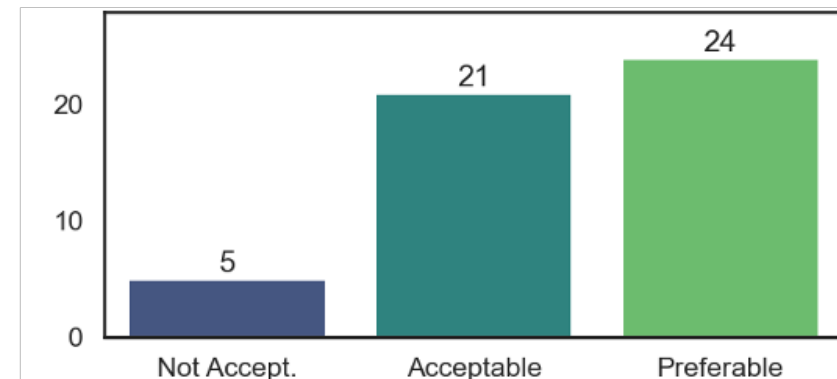
```
@Refinement("_ >= 0 && _ <= 100")  
public static int percentageFromGrade(@Refinement("grade >= 0") int grade,  
                                     @Refinement("scale > 0") int scale)
```

B

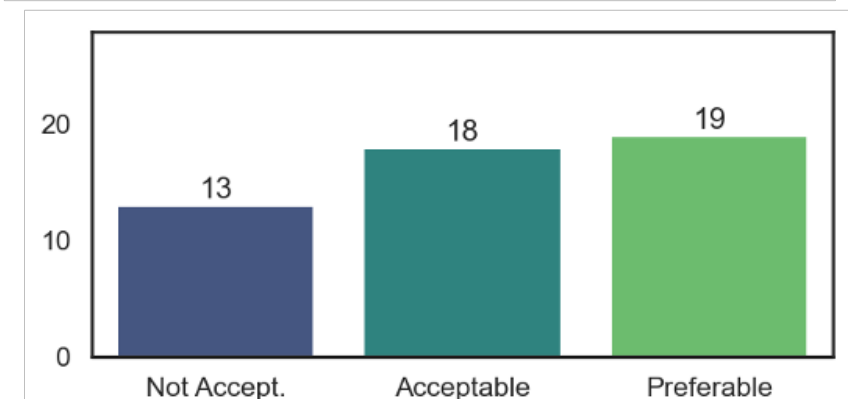
```
@Refinement("{grade >= 0} -> {scale > 0} -> {_ >= 0 && scale > 0}")  
public static int percentageFromGrade(int grade, int scale)
```



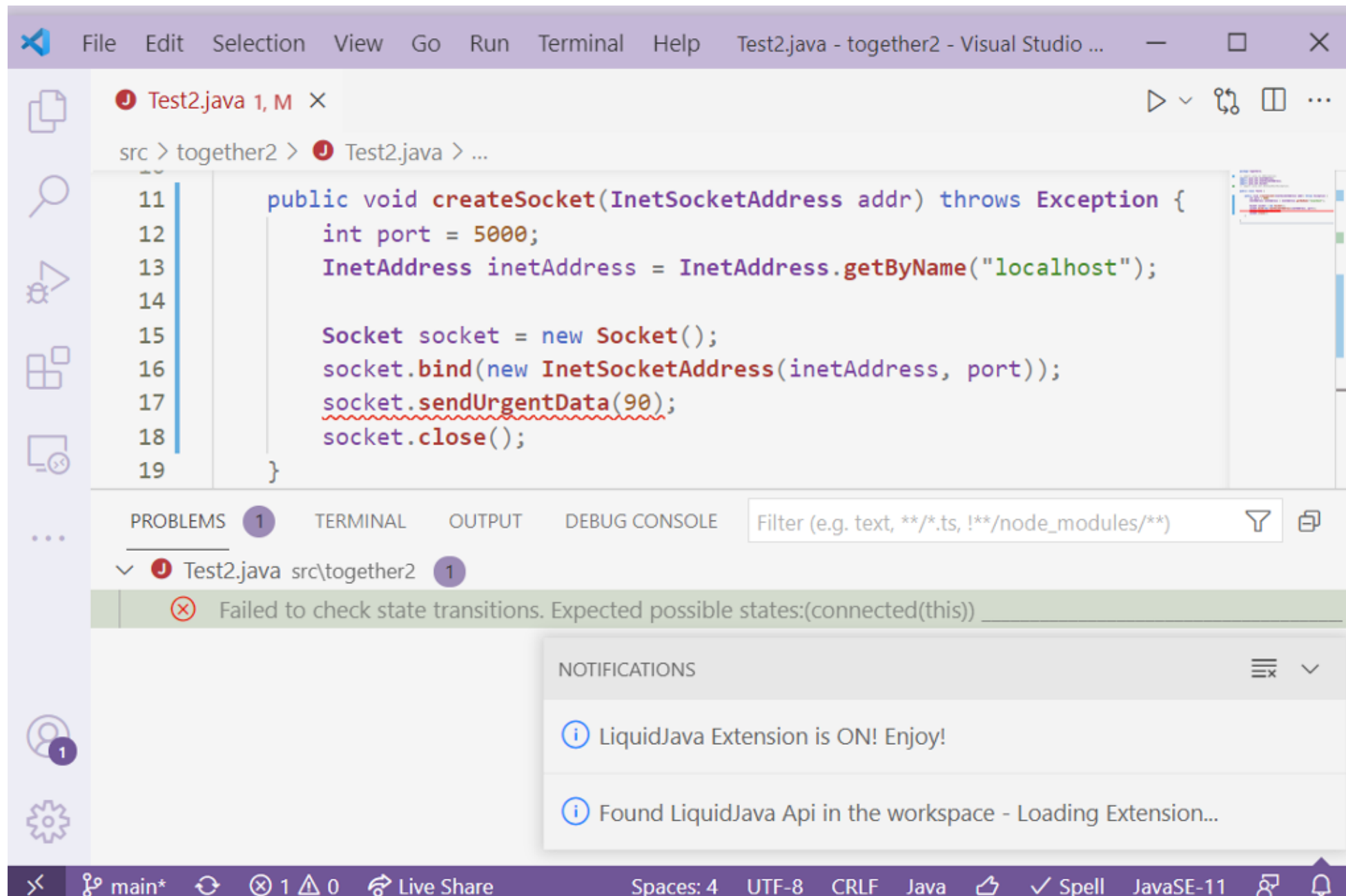
A



B

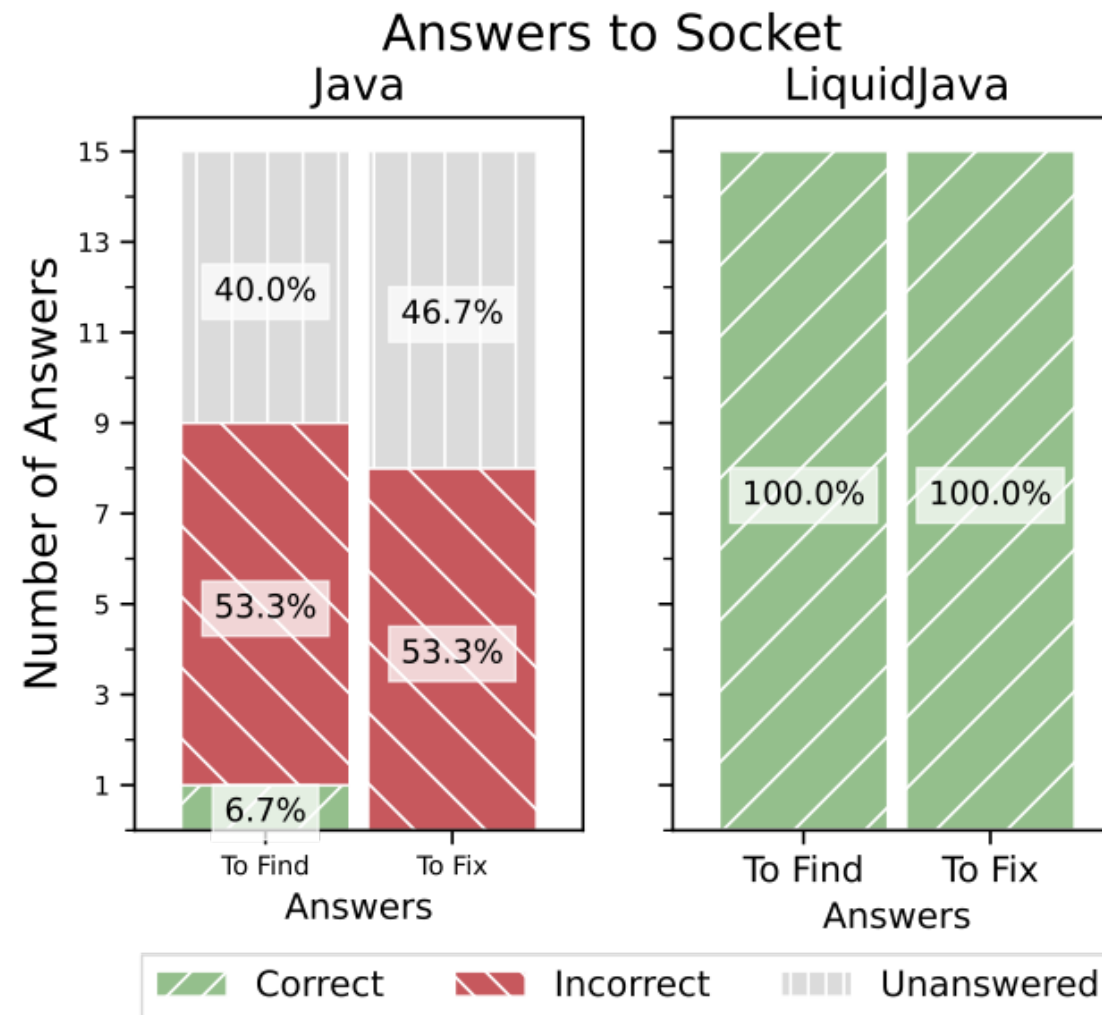


Editor Support for LiquidJava



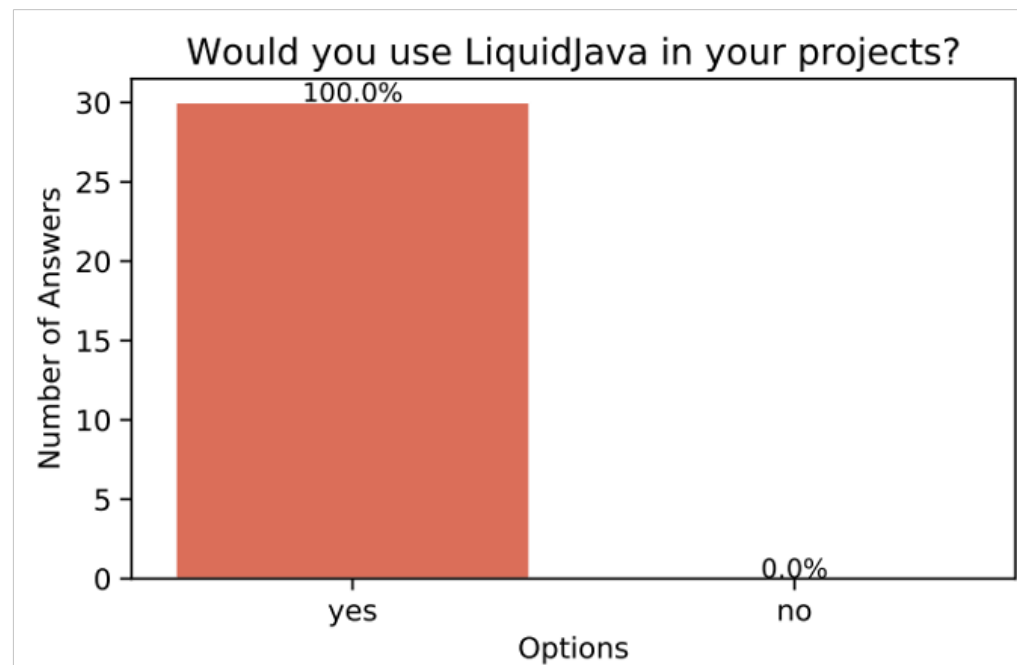
User Study on LiquidTypes

- ❑ LiquidJava **helped** developers find the bugs in code
- ❑ Best results for **lesser-known** classes



User Study on LiquidTypes

- ❑ Developers liked the **error reporting** and **state refinements** but want better better **plugin integration** and **error messages**
- ❑ Participants are **interested** in using LiquidJava



Liquid Types for Regression

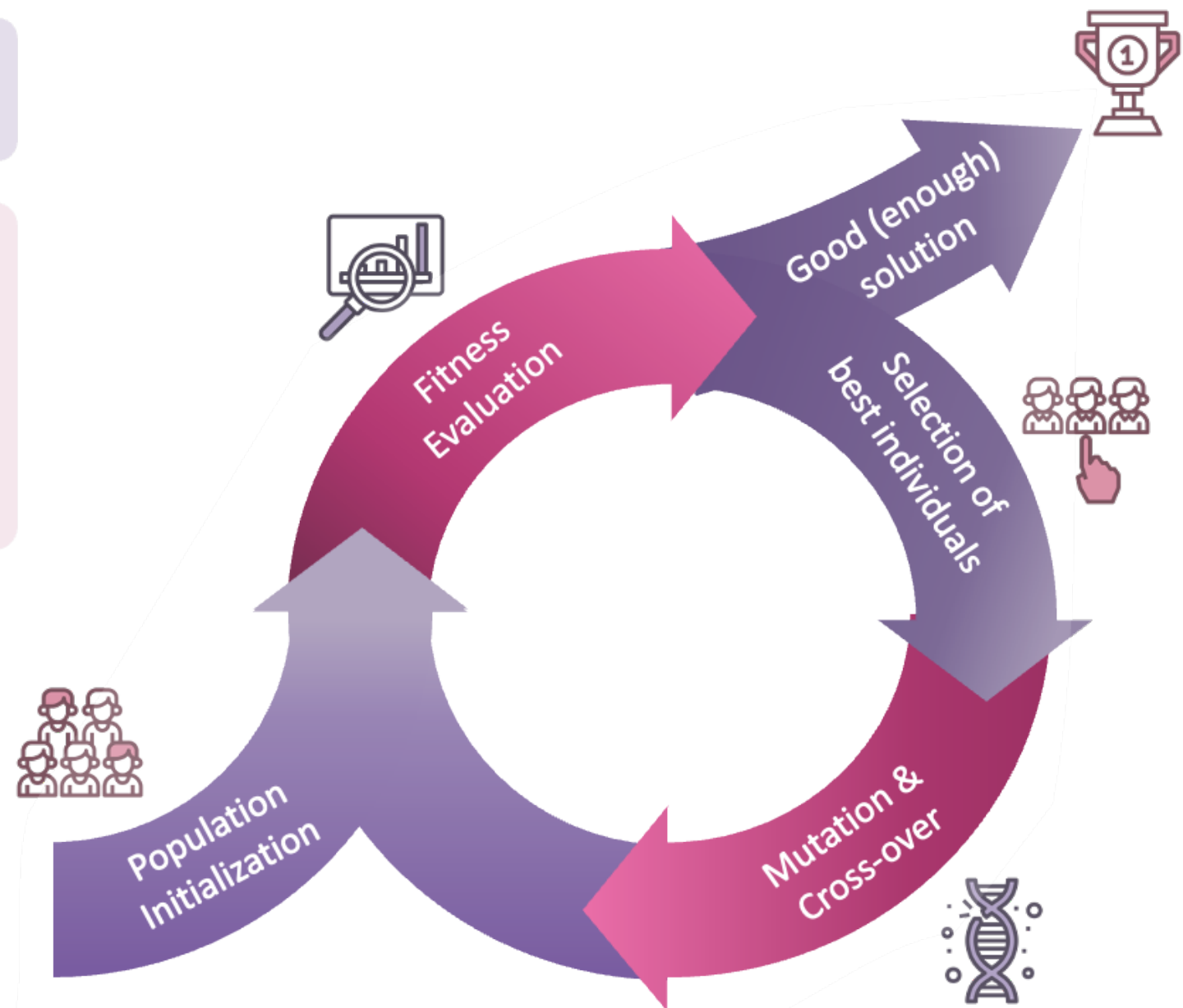
Program Synthesis

Program Synthesis with Refinements

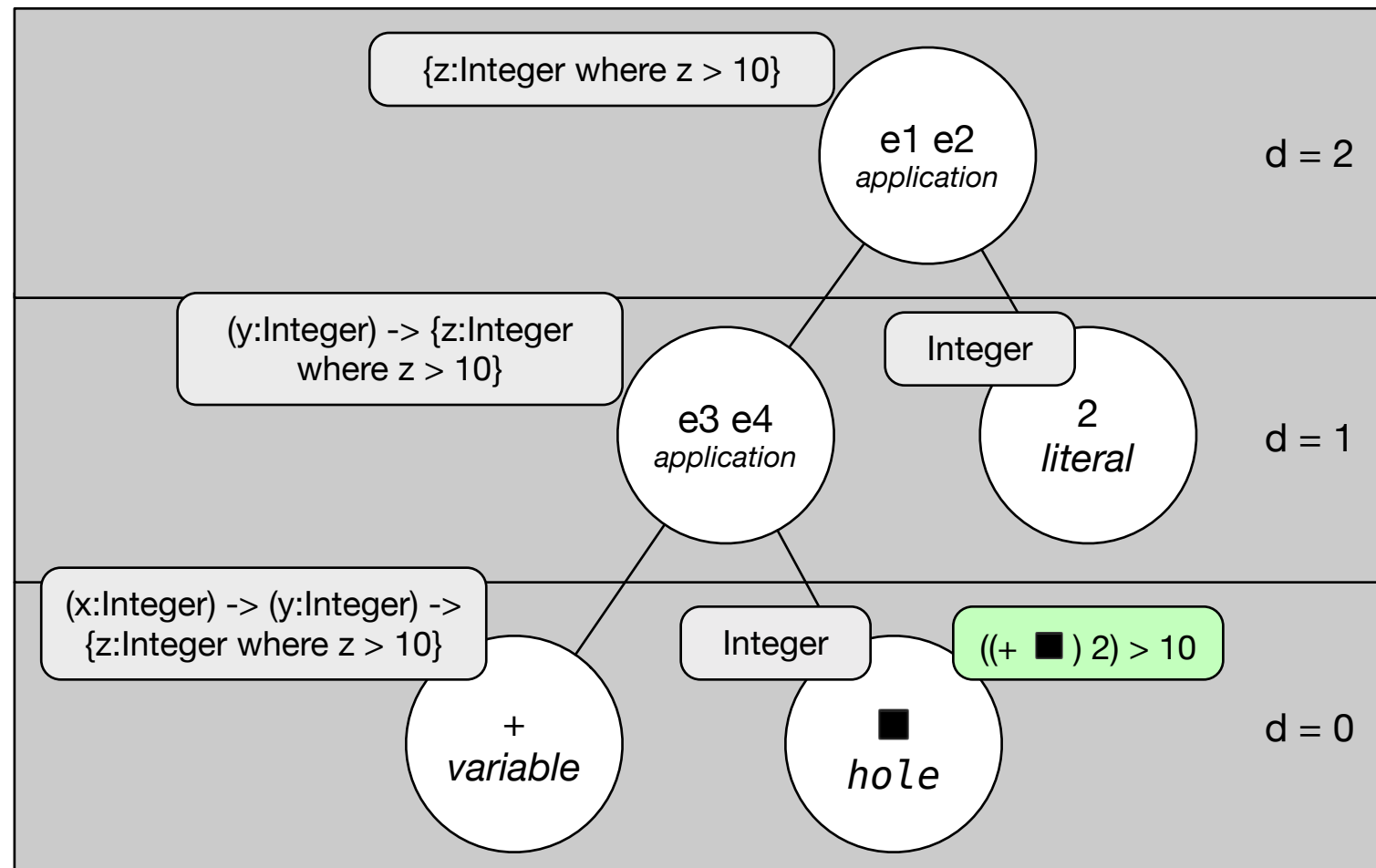
Traditional GP Symbolic Expression:

$2 * \text{temperature} * \text{Season}$

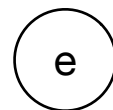
More Interpretable Expression:



Tree Generation



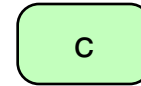
Legend



AST node



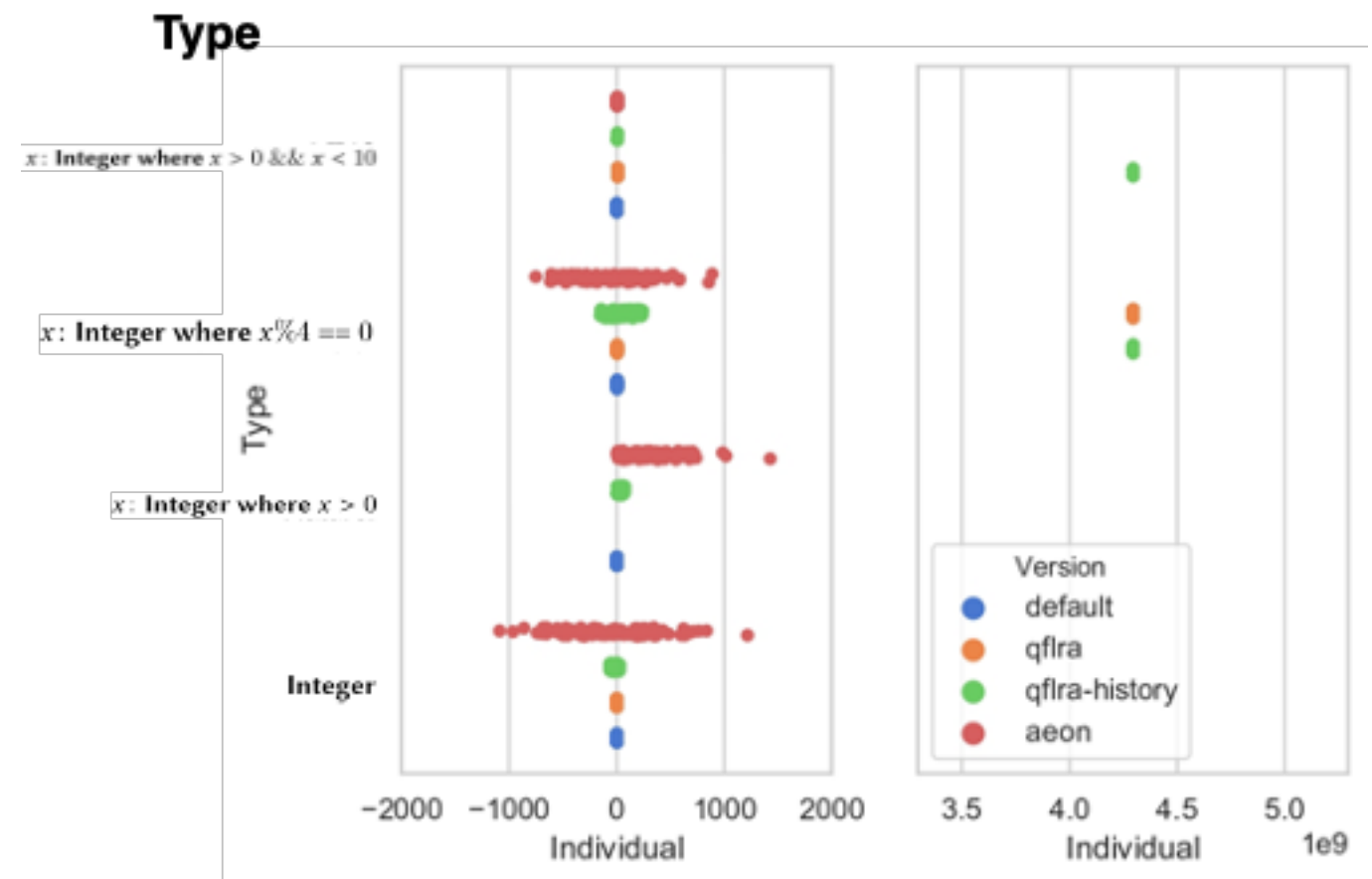
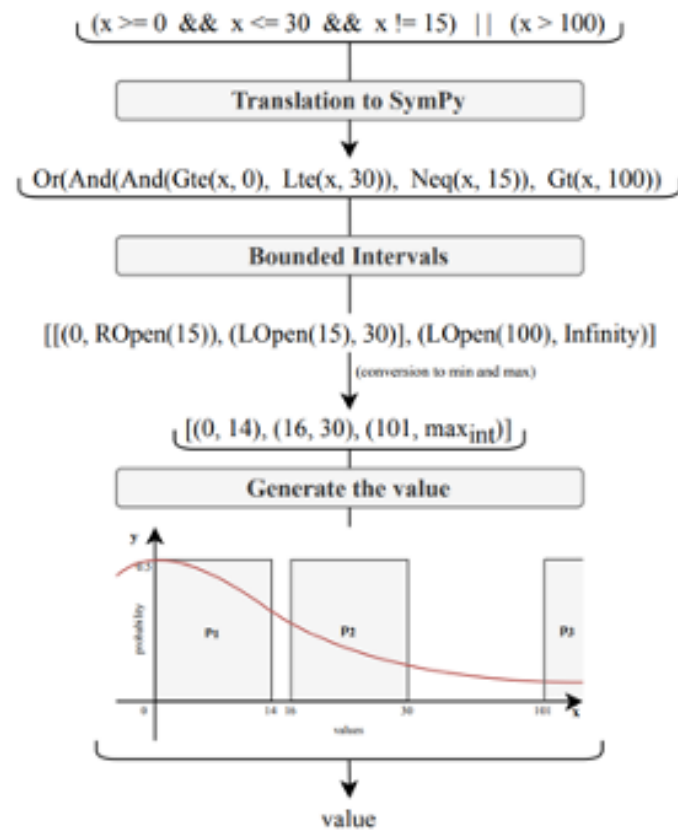
Expected Type



Expected condition

Problem: SMT Solvers lack consistent non-determinism

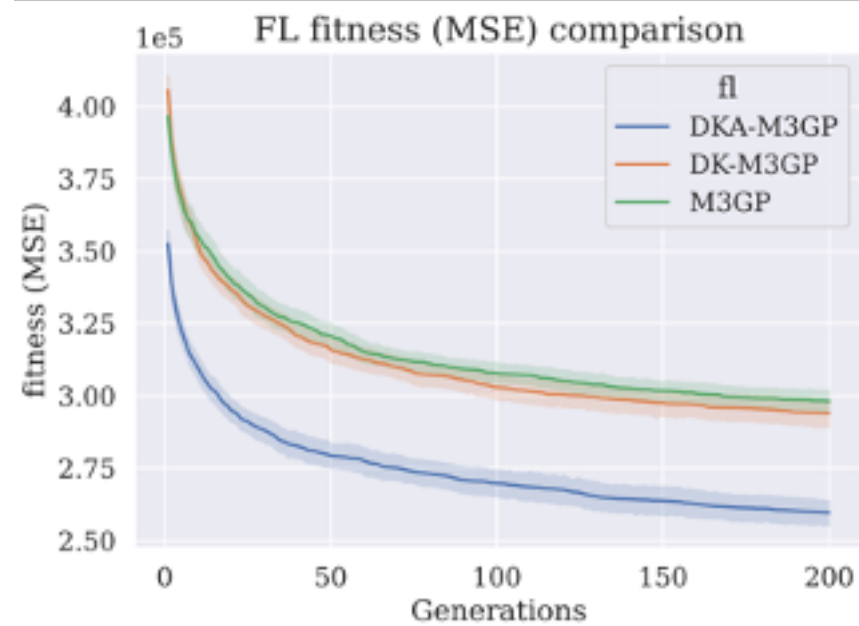
Solutions for type-guided sampling



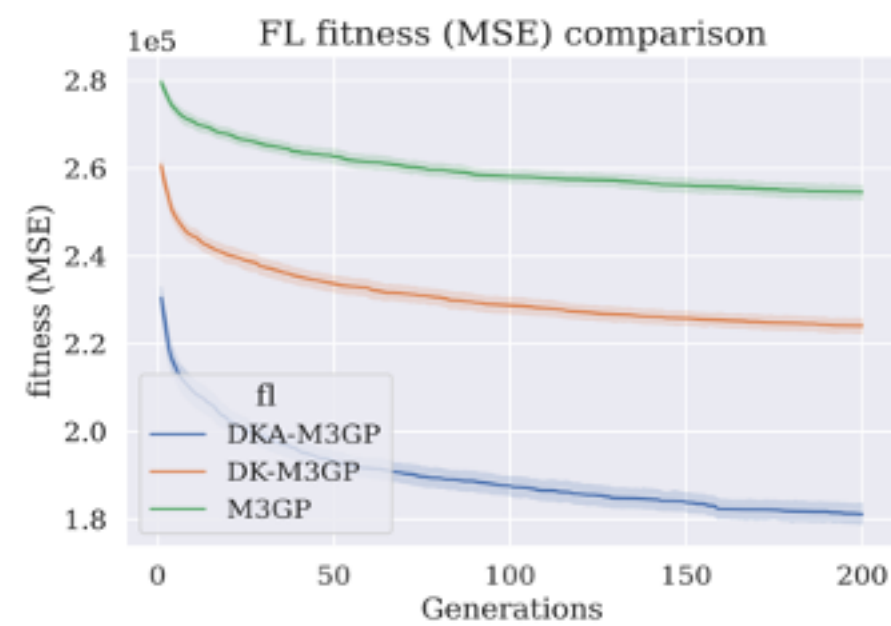
```
@weight(3)
@dataclass
class Container(Parent):
    age : Annotated[int, IntRange[0, 200]]
    height : Annotated[float, FloatRange[0.10, 2.5]]
    children : Annotated[List[Container], ListSizeBetween[1,30]]
    var_name : Annotated[str, VarRange[["x", "y", "z"]]]
```

Domain-Aware Feature Learning with Grammar-Guided Genetic Programming

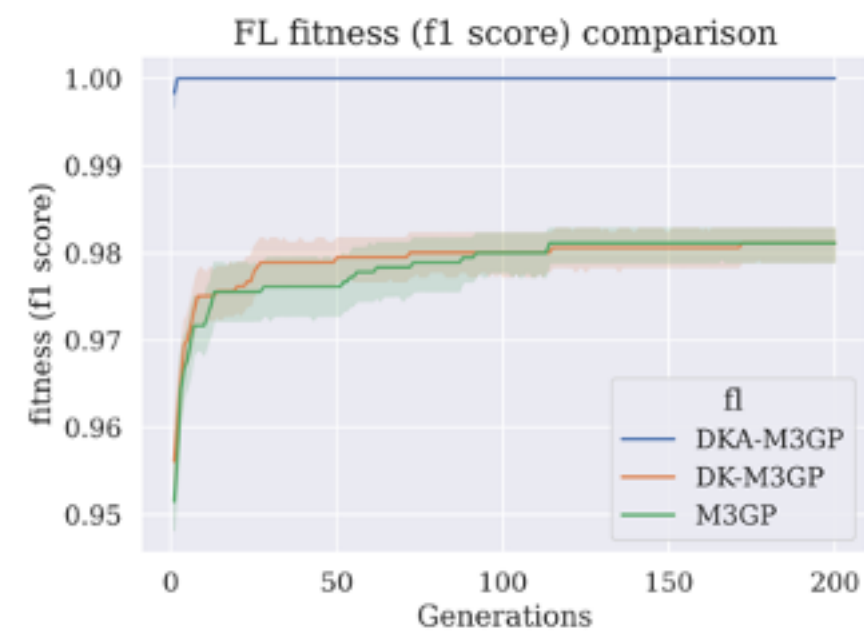
Boom bikes



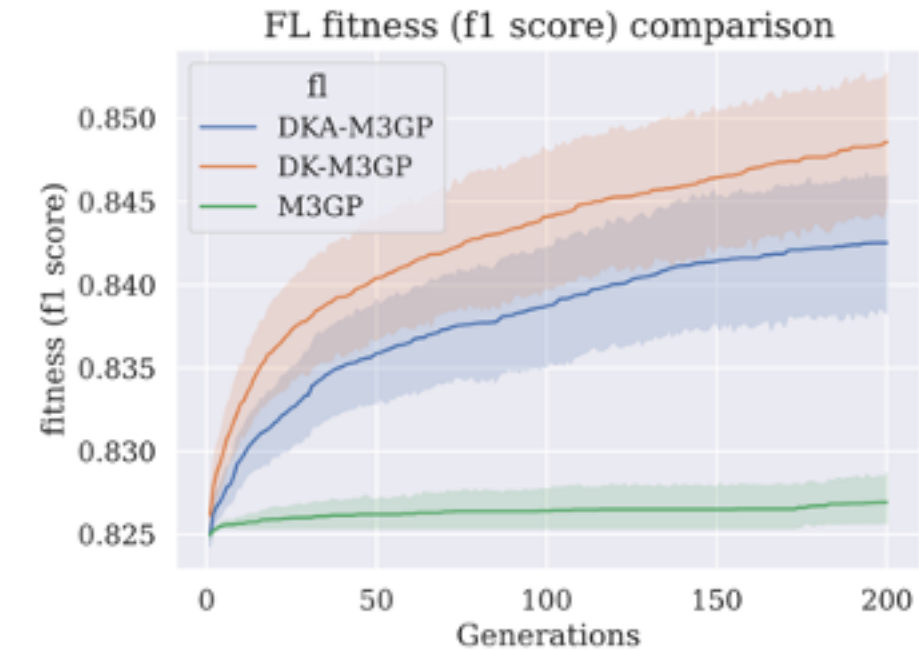
Website visitors



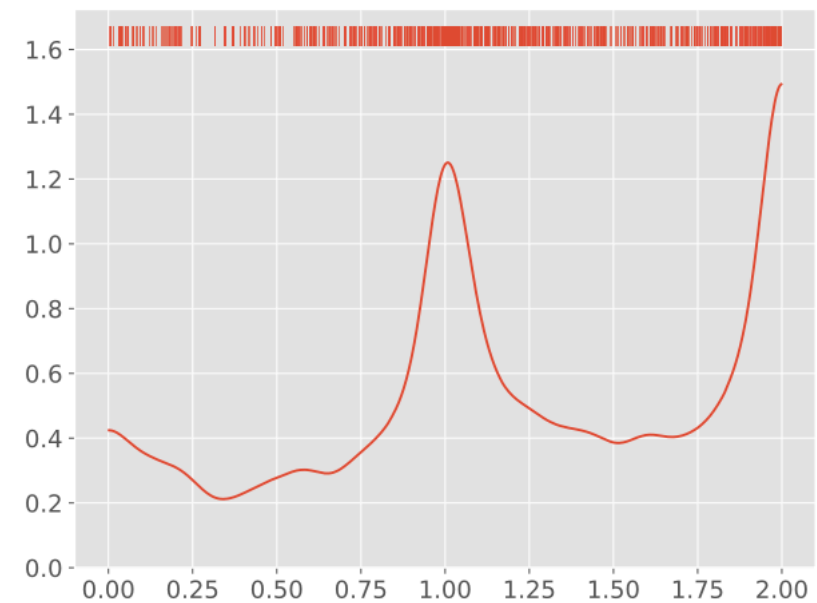
Caesarean



Credit Risk



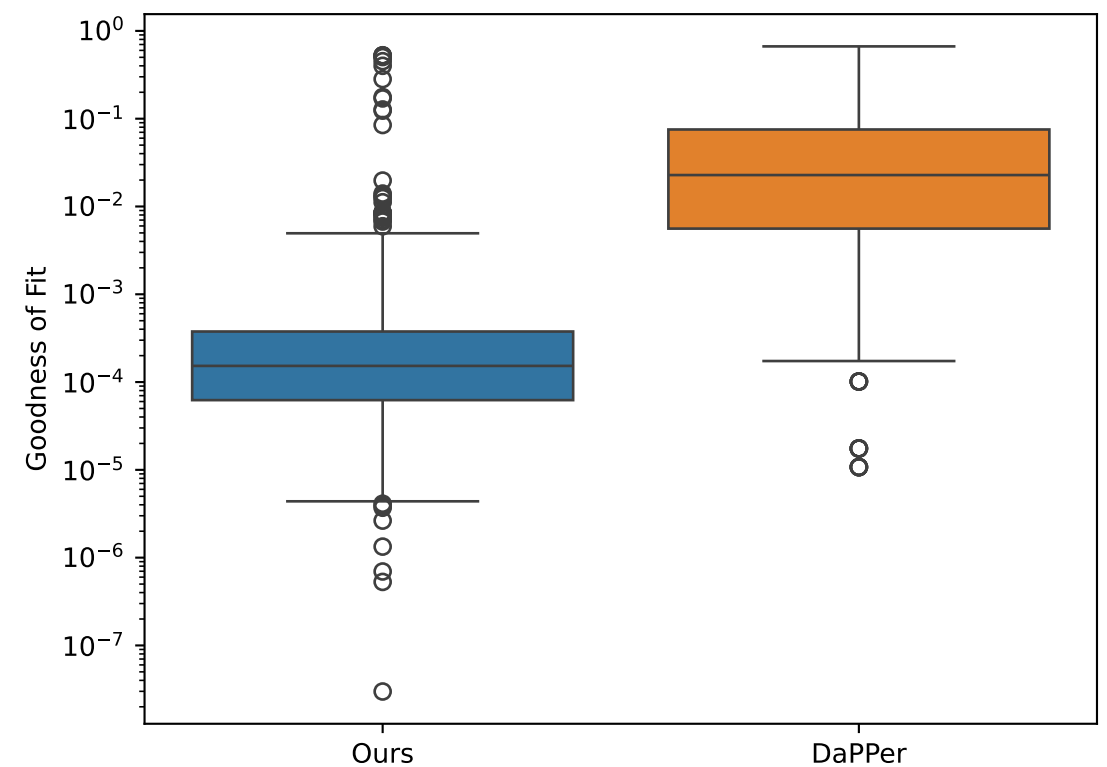
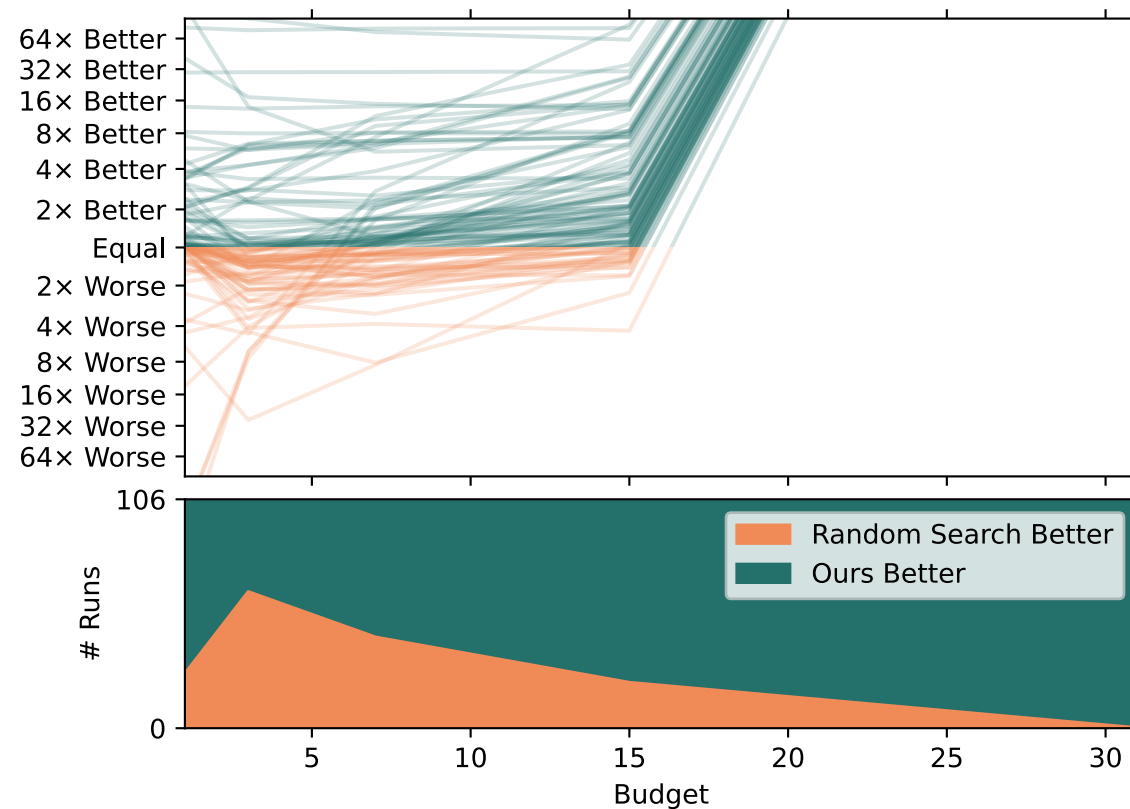
Probabilistic Symbolic Regression



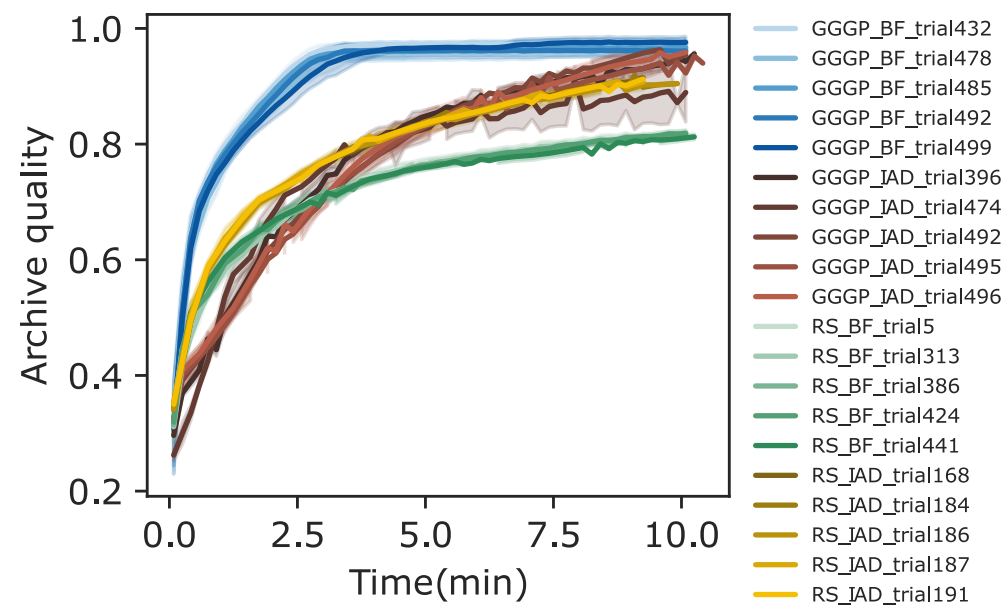
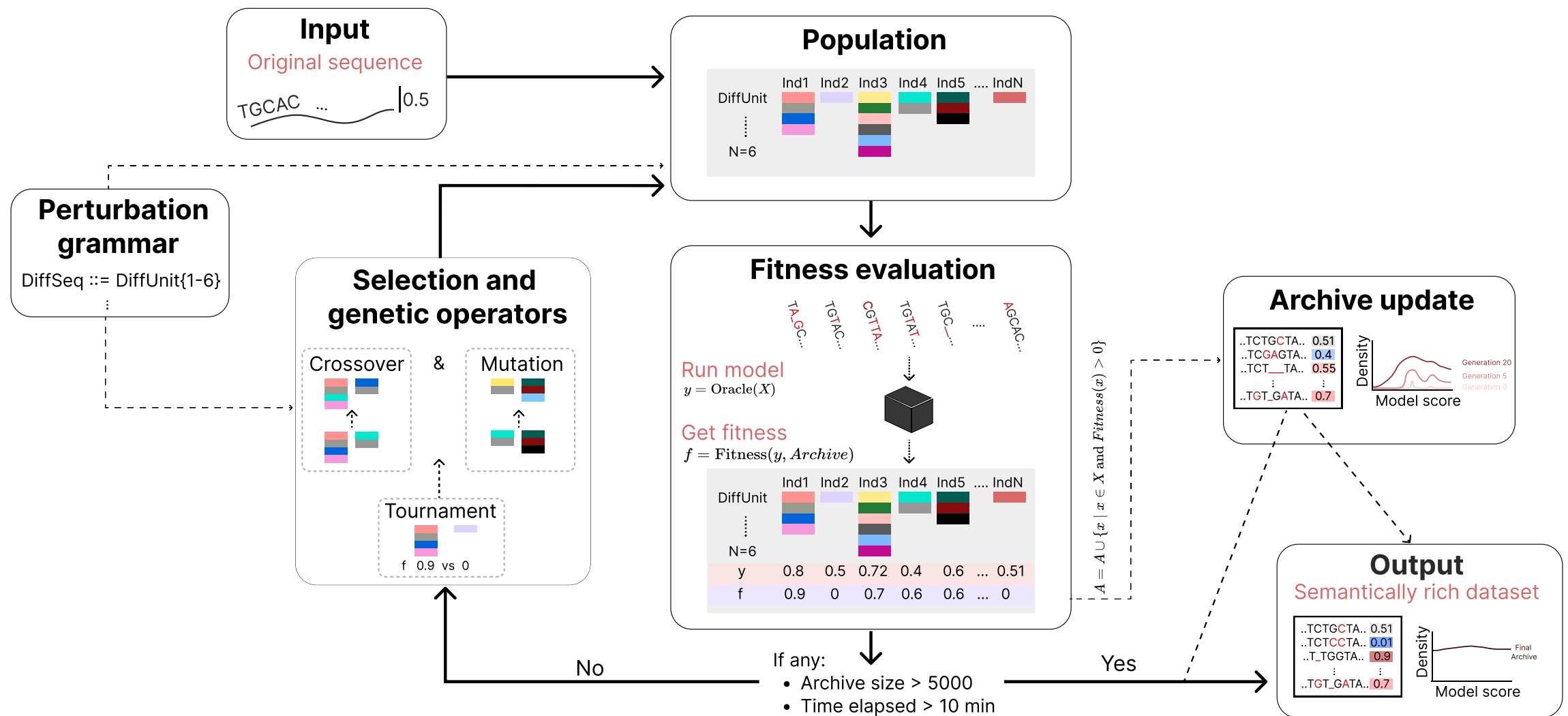
$x \sim \text{Beta}(0.3, 0.25)$

$y \sim \text{Beta}(0.4, 0.25)$

$x+y$



Semantically Rich Local Dataset Generation for Explainable AI in Genomics





Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl

Miles Cranmer^{1,2}

¹Princeton University, Princeton, NJ, USA

²Flatiron Institute, New York, NY, USA

May 2, 2023

PySR³ is an open-source library for practical *symbolic regression*, a type of machine learning which aims to discover human-interpretable symbolic models. PySR was developed to democratize and popularize symbolic regression for the sciences, and is built on a high-performance distributed backend, a flexible search algorithm, and interfaces with several deep learning packages. PySR's internal search algorithm is a multi-population evolutionary algorithm, which consists of a unique evolve-simplify-optimize loop, designed for optimization of unknown scalar constants in newly-discovered empirical expressions. PySR's backend is an extremely optimized Julia library SYMBOLICREGRESSION.JL⁴, which can be used directly from Julia. It is capable of fusing user-defined operators into SIMD kernels at runtime, performing automatic differentiation, and distributing populations of expressions to thousands of cores across a cluster. In describing this software, we also introduce a new benchmark, "*EmpiricalBench*," to quantify the applicability of symbolic regression algorithms in science. This benchmark measures recovery of historical empirical equations from original and synthetic datasets.



Shape-constrained multi-objective genetic programming for symbolic regression

C. Haider^{a,*}, F.O. de Franca^b, B. Burlacu^a, G. Kronberger^a

^aJosef Ressel Center for Symbolic Regression, University of Applied Sciences Upper Austria, Softwarepark 11, Hagenberg, 4232, Austria

^bCenter for Mathematics, Computation and Cognition (CMCC), Heuristics, Analysis and Learning Laboratory (HAL), Federal University of ABC, Santo André, Brazil

ARTICLE INFO

Article history:
Received 15 March 2022
Received in revised form 10 October 2022
Accepted 21 November 2022
Available online 24 November 2022

Keywords:
Genetic Programming
Multi-objective optimization
Shape-constrained regression
Symbolic regression

ABSTRACT

We describe and analyze algorithms for shape-constrained symbolic regression, which allow the inclusion of prior knowledge about the shape of the regression function. This is relevant in many areas of engineering — in particular, when data-driven models, which are based on data of measurements must exhibit certain properties (e.g. positivity, monotonicity, or convexity/concavity). To satisfy these properties, we have extended multi-objective algorithms with shape constraints. A soft-penalty approach is used to minimize both the constraint violations and the prediction error. We use the non-dominated sorting genetic algorithm (NSGA-II) as well as the multi-objective evolutionary algorithm based on decomposition (MOEA/D). The algorithms are tested on a set of models from physics textbooks and compared against previous results achieved with single objective algorithms. Further, we generated out-of-domain samples to test the extrapolation behavior using shape constraints and added a different level of noise on the training data to verify if shape constraints can still help maintain the prediction errors to a minimum and generate valid models. The results showed that the multi-objective algorithms were capable of finding mostly valid models, also when using a soft-penalty approach. Further, we investigated that NSGA-II achieved the best overall ranks on high noise instances.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

6. Analysis of In-domain performance

6.1. Problem instances

We use a subset of problem instances from the *Feynman Symbolic Regression Database* [40] which is a collection of models from physics textbooks. We select instances based on the difficulty reported in [40], whereby we kept only the harder instances² for which we could easily derive monotonicity constraints. The selected functions are shown in Table 2. They are smooth, non-linear, and have only a few inputs. Although we are using synthetic data, these data are related to physical laws and therefore similar data could be measured in real world problems.

For each instance, we defined several shape constraints derived from the known expression. The shape constraints are shown in Table 3 whereby we use a compact tuple notation. The first element of the tuple is the allowed range for model output and the remaining elements indicate for each input variable whether the partial derivative of the model over that variable is non-positive (−1, monotone decreasing) or non-negative (1, monotone increasing). The value zero indicates no constraint on the partial derivative.

Full Dependent Types

Full Dependent Types

Often found in Interactive Theorem Provers (ITP)

- They require writing the proof manually.
- They are very expressive.

```
theorem proof :  $\forall v, v > 0 \rightarrow v \neq 0 :=$  by  
  intro v  
  intro vgt0  
  induction v  
  simp at vgt0  
  simp
```

INVITED ADDRESS



AMS Colloquium Lectures: Lecture I



Machine Assisted Proof

Terence Tao, *University of California, Los Angeles*

Introduced by **Bryna Kra**, *Northwestern University*



Code - InsidersFileEditSelectionViewGoRunTerminalWindowHelp

lean4-example [SSH: dgx]

Lean4Example.lean

Lean4Example.lean > add_abc

```
1 -- Lean Copilot is open-sourced at https://github.com/lean-dojo/LeanCopilot
2 import LeanCopilot
3 import Mathlib.Data.Set.Basic
4
5
6 theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
7   intro a b c
8   apply Nat.add_right_comm
9
10
11   by tac constructs a term of the expected type by running the tactic(s) tac .
12
13 text x
14   simp_all only [Set.mem_inter_iff]
15   apply Iff.intro
16     · intro a
17       simp_all only [and_self]
18     · intro a
19       simp_all only [and_self]
```

Lean Infoview

Lean4Example.lean:8:26

Tactic state

No goals.

Expected type

$$a\ b\ c : \mathbb{N}$$
$$\vdash \forall (n\ m\ k : \mathbb{N}), n + m + k = n + k + m$$

All Messages (0)

SSH: dgxml4tp-tutorial*000

Next Steps

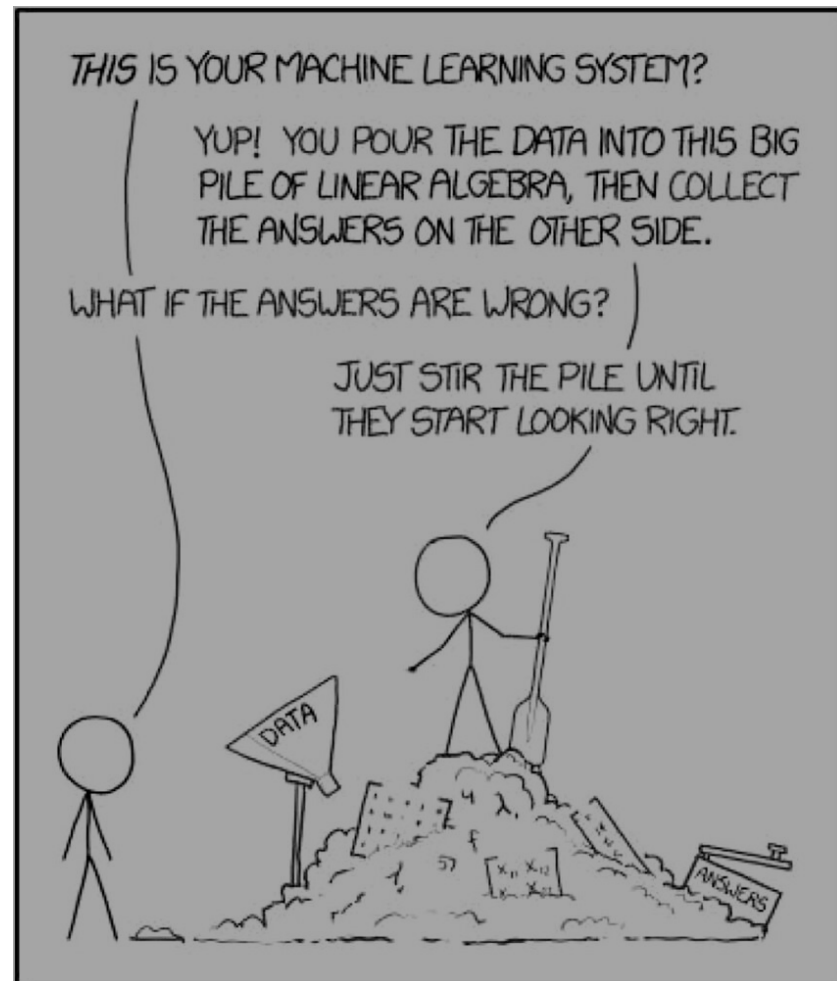


Curry–Howard correspondence

- Use Liquid Types to synthesize Mathematical Proofs
- Unlike with LLMs, any generated program is guaranteed to be correct!

Questions?

Examples:



Can we use Genetic Programming to learn new laws, from observational data?

Is Genetic Programming better than Deep Learning?

Should we formalize our proofs in Lean?

Is there a physics library for Lean?