LExecutor: Learning Guided Execution

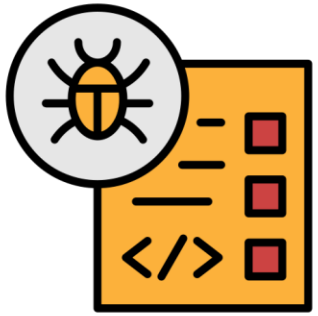
Beatriz Souza and Michael Pradel

European Research Council

Established by the European Commission

Why Execute Code?

- Enables various dynamic analysis



Bug detection

Vulnerability detection

Semantic equivalence

Performance analysis

Motivation

```
if (not has_min_size(all_data)):
    raise RuntimeError("not enough data")

train_len = round(0.8 * len(all_data))

logger.info(f"Extracting training data with {config_str}")

train_data = all_data[0:train_len]

# ...
```

Motivation

Missing variable

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

Motivation

Missing function

Missing variable

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

Motivation

Missing function

Missing variable

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

Missing variable

Motivation

Missing function

Missing variable

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

Missing variable

Missing import
and attribute

Executing is **NOT** Easy

Incomplete code occurs in many usage scenarios:

- Code snippets from **Stack Overflow**
- Code generated by **language models**
- Code extracted from **complex projects**

Executing is **NOT** Easy

Incomplete code occurs in many usage scenarios:

- Code snippets from **Stack Overflow**
- Code generated by **language models**
- Code extracted from **complex projects**

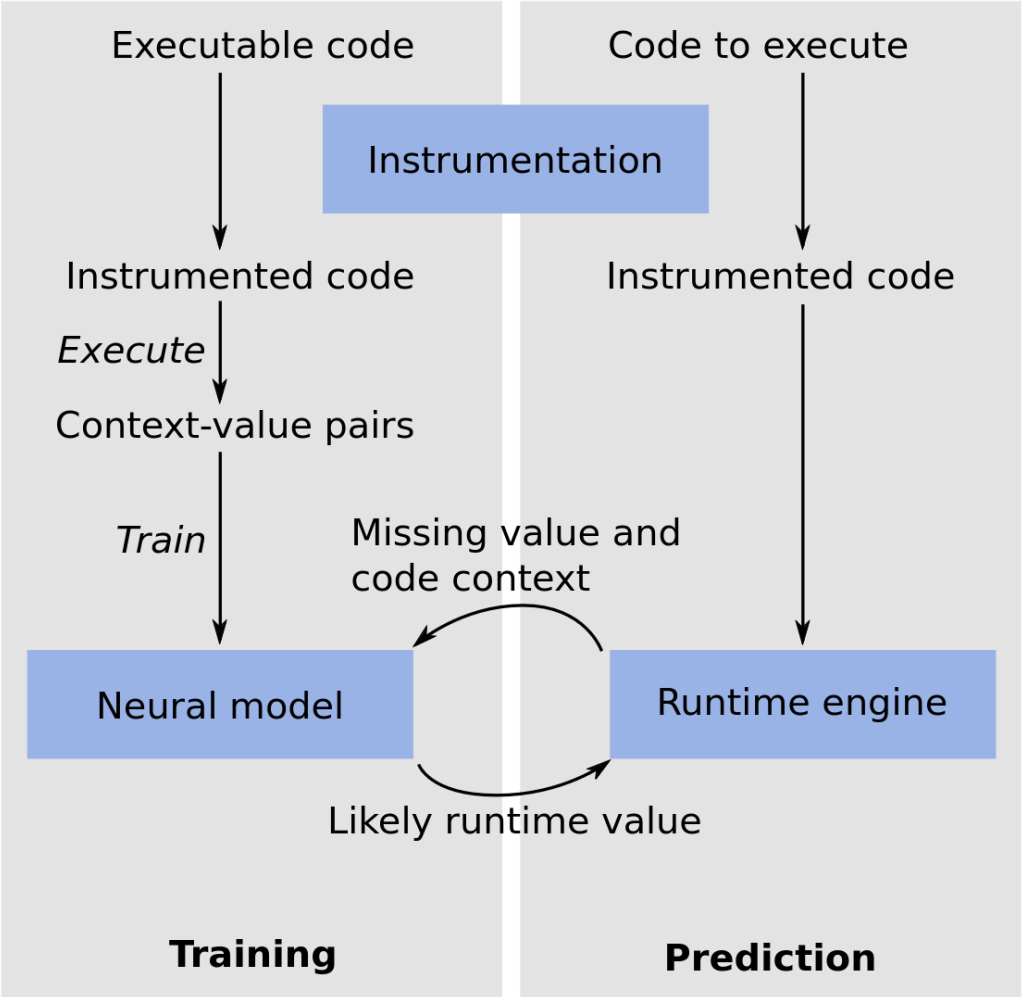
Can we automatically fill in the missing information?

LExecutor

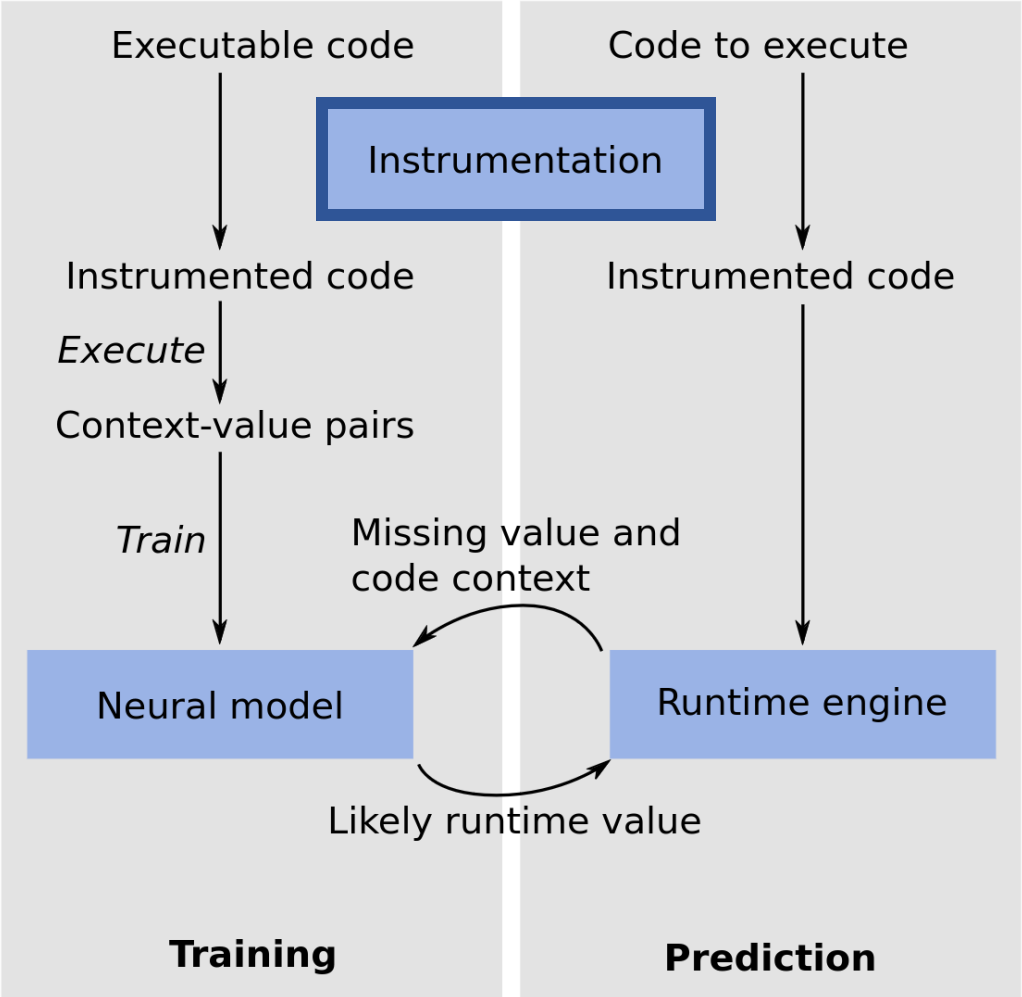
Learning-guided approach for **executing arbitrary code snippets**

- Predict missing values with neural model
- Inject values into the execution

Overview of LExecutor



Overview of LExecutor



Instrumentation

AST-based code transformations:

- Wrap variable reads into _n_()
- Wrap attribute reads into _a_()
- Wrap calls of functions and methods into _c_()

Instrumentation

AST-based code transformations:

- Wrap variable reads into _n_()
- Wrap attribute reads into _a_()
- Wrap calls of functions and methods into _c_()

Original code:

```
y = x + 1
```

Instrumented code:

```
y = _n_(537, "x", lambda: x) + 1
```

Instrumentation

AST-based code transformations:

- Wrap variable reads into _n_()
- Wrap attribute reads into _a_()
- Wrap calls of functions and methods into _c_()

Original code:

y = **x** + 1

Instrumented code:

y = _n_(537, "x", lambda: x) + 1

Variable name

Instrumentation

AST-based code transformations:

- Wrap variable reads into _n_()
- Wrap attribute reads into _a_()
- Wrap calls of functions and methods into _c_()

Original code:

y = **x** + 1

Instrumented code:

y = _n_(537, "x", lambda: x) + 1

Variable name

Function to read value and react in a controlled manner

Instrumentation

AST-based code transformations:

- Wrap variable reads into _n_()
- Wrap attribute reads into _a_()
- Wrap calls of functions and methods into _c_()

Original code:

y = **x** + 1

Instrumented code:

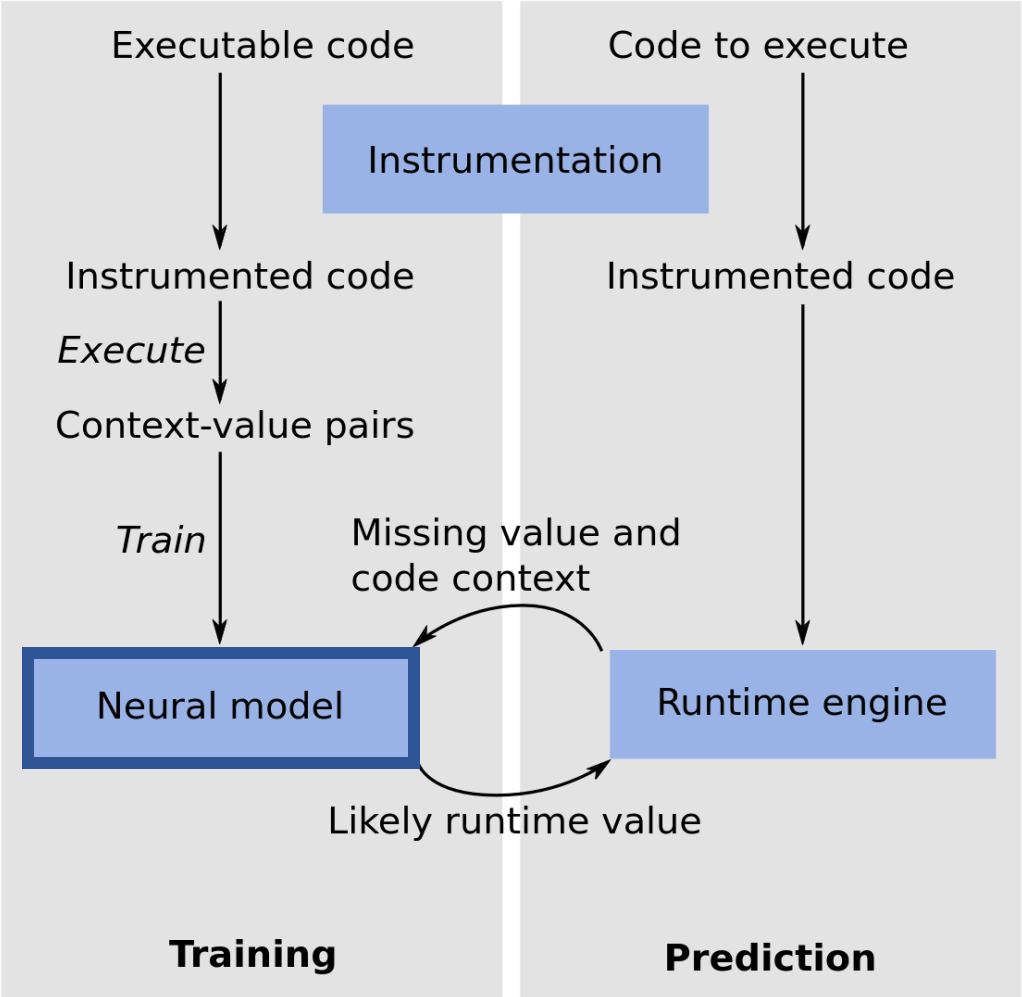
y = _n_(537, "x", lambda: x) + 1

Unique ID

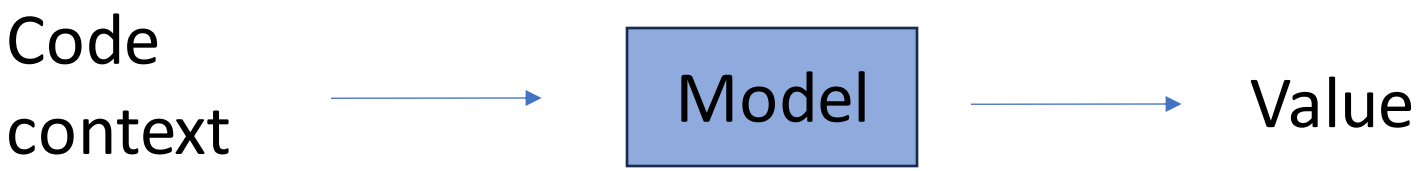
Variable name

Function to read value and react in a controlled manner

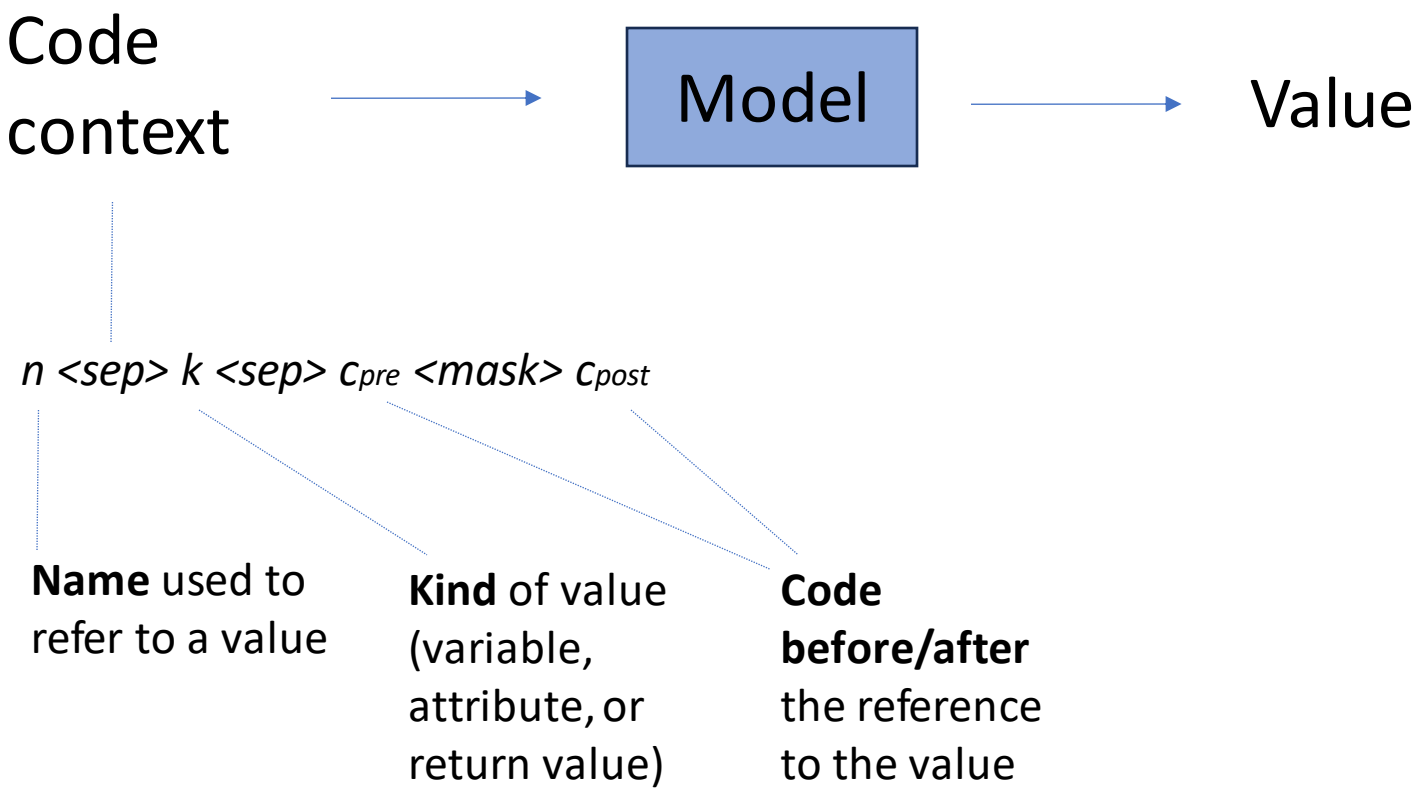
Overview of LExecutor



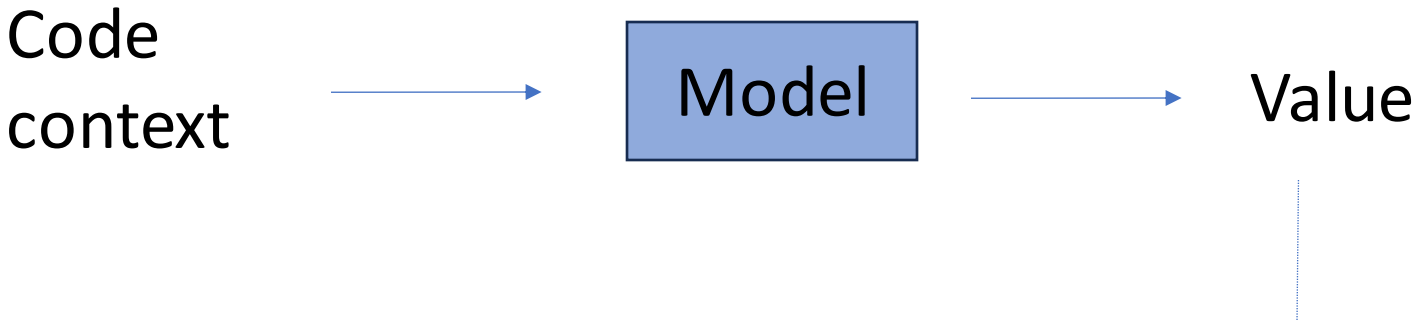
Neural Model: Data Representation



Neural Model: Data Representation



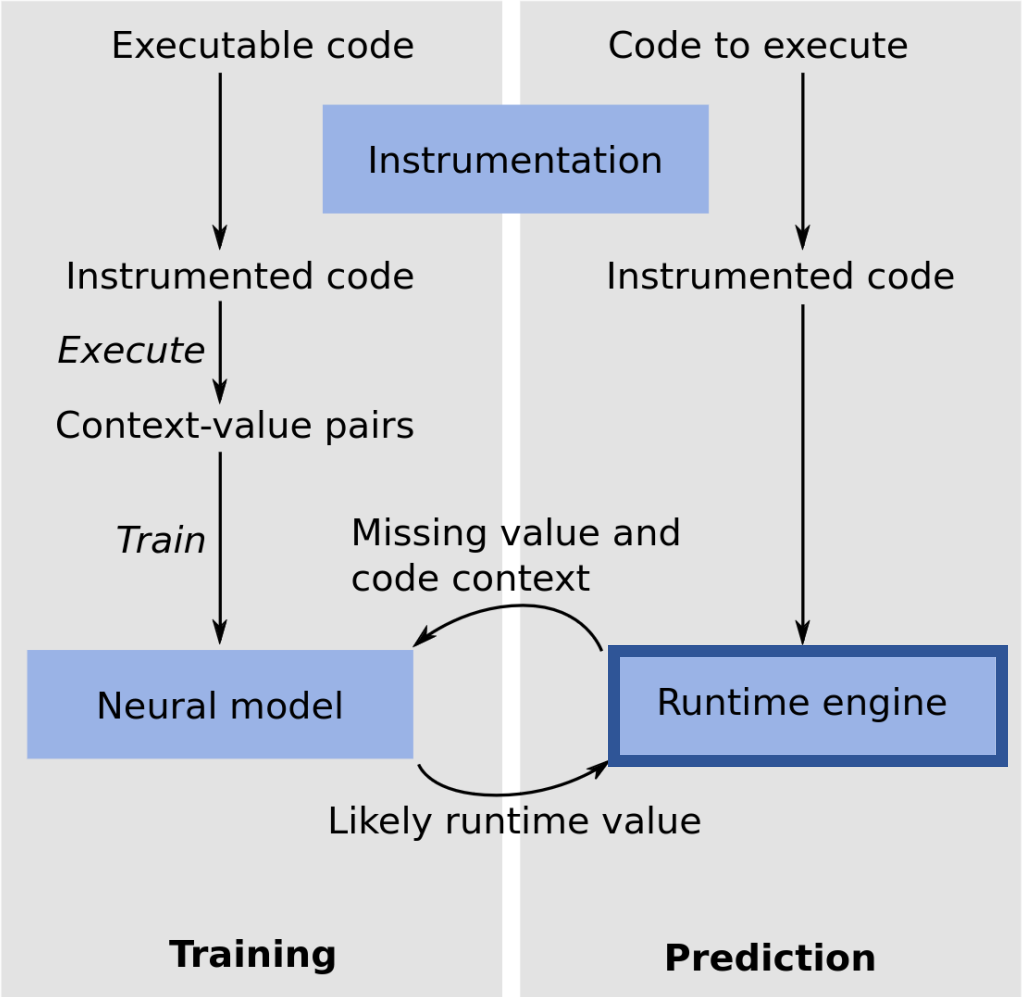
Neural Model: Data Representation



Concrete values **abstracted** into 23 classes, e.g.,

- None, True, False
- Negative/zero/positive integer
- Empty/non-empty list
- Callable
- …

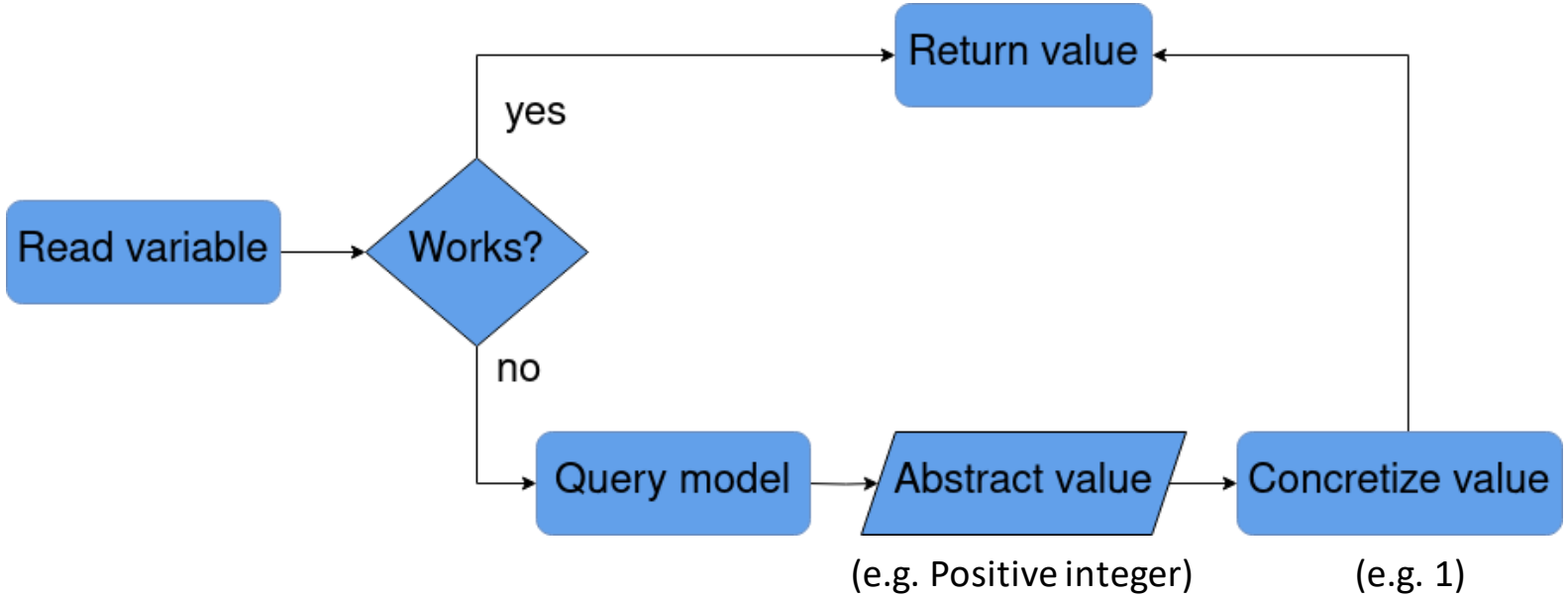
Overview of LExecutor



Runtime Engine

During prediction:

For each use of a value



Evaluation

- RQ1: Accuracy of the Neural Model
- RQ2: Effectiveness at Covering Code
- RQ3: Efficiency at Guiding Executions
- RQ4: Using LExecutor to Find Semantics-Changing Commits

Evaluation

- **RQ1: Accuracy of the Neural Model**
- **RQ2: Effectiveness at Covering Code**
- RQ3: Efficiency at Guiding Executions
- RQ4: Using LExecutor to Find Semantics-Changing Commits

RQ1: Accuracy of the Neural Model

RQ1: Accuracy of the Neural Model

Models:

- CodeT5
- CodeBERT

RQ1: Accuracy of the Neural Model

Datasets:

Project	Description	Unique value-use events
Ansible	Automation of software infrastructure	43,090
Django	Web framework	121,567
Keras	Deep learning library	30,709
Request	Client-side HTTP library	5,273
Rich	Text formatting in the terminal	25,370
Total		226,009

RQ1: Accuracy of the Neural Model

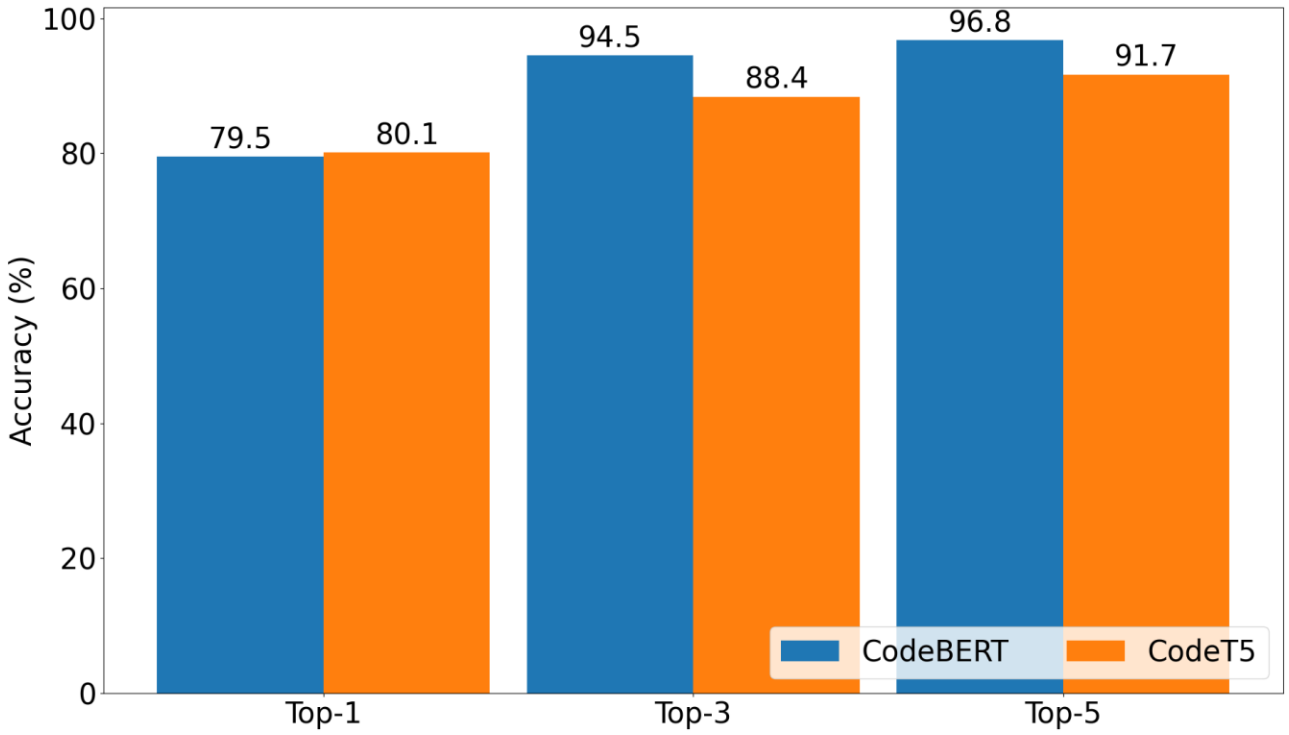
Datasets:

Project	Description	Unique value-use events
Ansible	Automation of software infrastructure	43,090
Django	Web framework	121,567
Keras	Deep learning library	30,709
Request	Client-side HTTP library	5,273
Rich	Text formatting in the terminal	25,370
Total		226,009

95%: Training

5%: Testing

RQ1: Accuracy of the Neural Model



RQ2: Effectiveness at Covering Code

RQ2: Effectiveness at Covering Code

Datasets:

Functions

Project	Description	Functions	LoC
Black	Code formatting	200	2,961
Flask	Web applications	200	1,354
Pandas	Data analysis	200	2,015
Scrapy	Web scraping	200	1,198
TensorFlow	Deep learning	200	2,125
Total		1,000	9,653

RQ2: Effectiveness at Covering Code

Datasets:

Functions

Project	Description	Functions	LoC
Black	Code formatting	200	2,961
Flask	Web applications	200	1,354
Pandas	Data analysis	200	2,015
Scrapy	Web scraping	200	1,198
TensorFlow	Deep learning	200	2,125
Total		1,000	9,653

stackoverflow **Snippets**

462 syntactically correct code snippets in answers to 1,000 Python-related questions

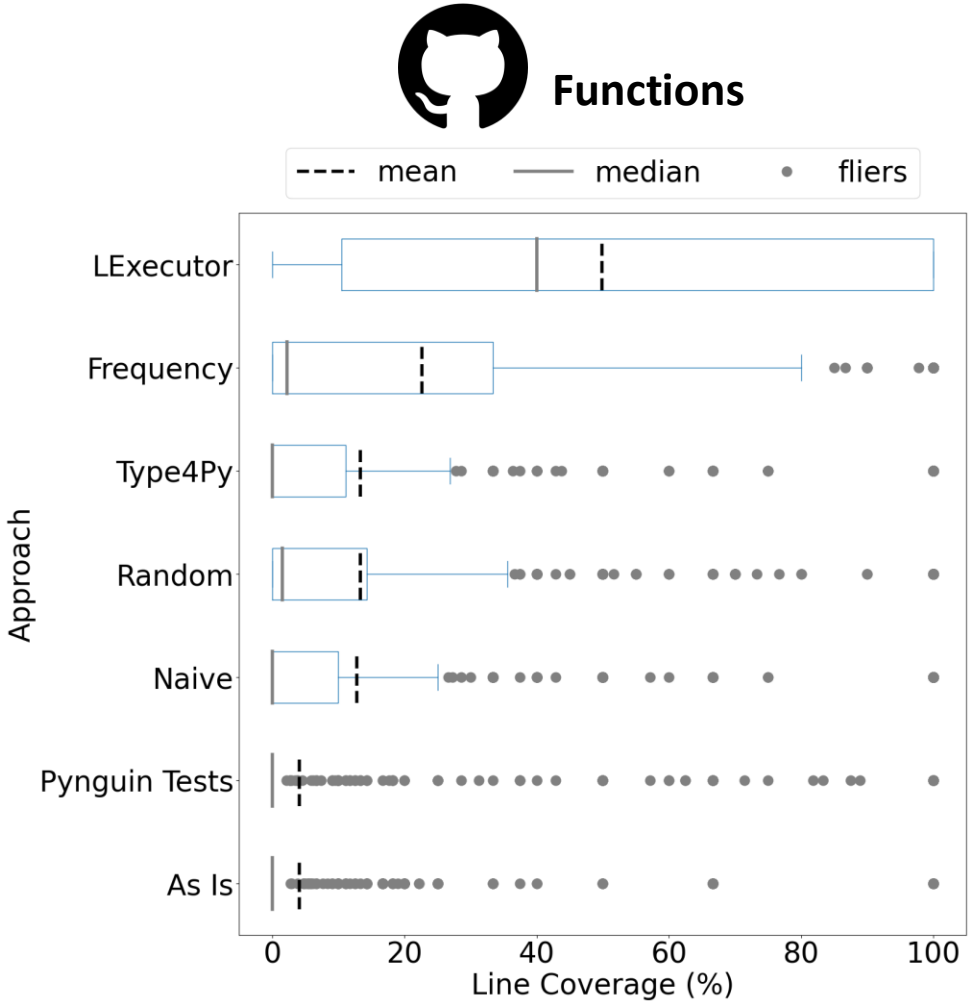
RQ2: Effectiveness at Covering Code

Baselines:

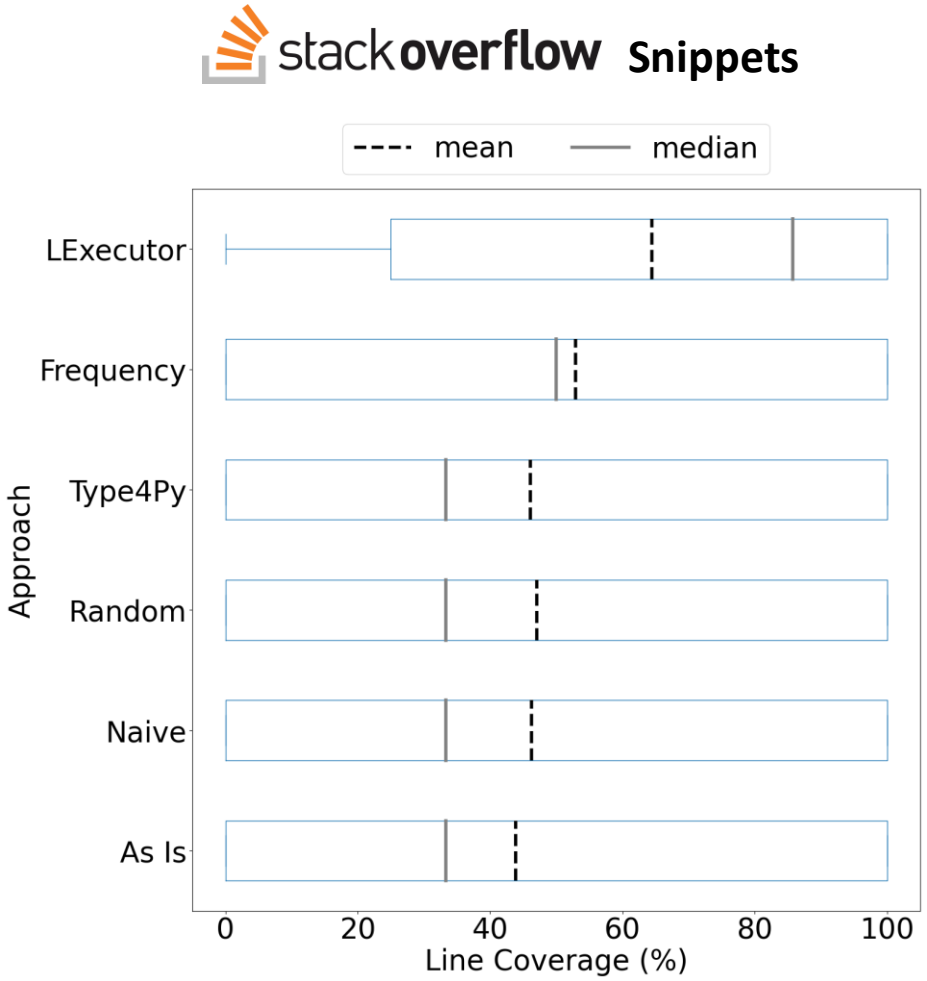
- As Is
- Naive
- Random
- Frequency
- Type4Py [1]
- Pynguin Tests [2]

1. Type4Py: Practical deep similarity learning-based type inference for Python, ICSE'22 (Amir M Mir, Evaldas Latoškinas, Sebastian Proksch, and Georgios Gousios)
2. Automated Unit Test Generation for Python, SSBSE'20 (Stephan Lukasczyk, Florian Kroiß, and Gordon Fraser)

RQ2: Effectiveness at Covering Code



RQ2: Effectiveness at Covering Code



LExecuting the Motivating Example

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

LExecuting the Motivating Example

Non-empty list

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

LExecuting the Motivating Example

Function that returns True

Non-empty list

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

LExecuting the Motivating Example

Function that returns True

Non-empty list

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

Non-empty string

LExecuting the Motivating Example

Function that returns True

Non-empty list

```
1   if (not has_min_size(all_data)):
2       raise RuntimeError("not enough data")
3
4   train_len = round(0.8 * len(all_data))
5
6   logger.info(f"Extracting training data with {config_str}")
7
8   train_data = all_data[0:train_len]
9
10  # ...
```

Non-empty string

Object with a method

Why Execute Code?

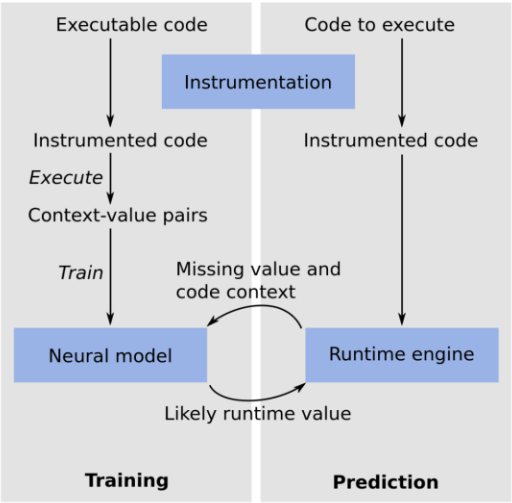
- Enables various dynamic analysis

Executing is **NOT** Easy

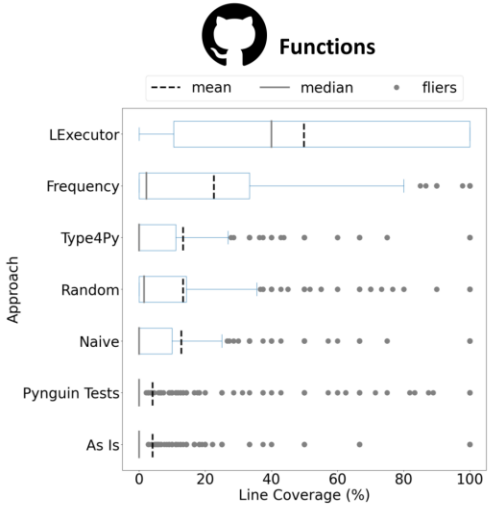
There are many **incomplete code**:

- Code snippets from **Stack Overflow**
- Code generated by **language models**
- Code extracted from **complex projects**

Overview of LExecutor



RQ2: Effectiveness at Covering Code



Abstraction of Values

Table 1: Fine-grained abstraction and concretization of values.

Abstract class of values	Concretization (Python)
Common primitive values:	
None	`None`
True	`True`
False	`False`
Built-in numeric types:	
Negative integer	`-1`
Zero integer	`0`
Positive integer	`1`
Negative float	`-1.0`
Zero float	`0.0`
Positive float	`1.0`
Strings:	
Empty string	`""`
Non-empty string	`"a"`
Built-in sequence types:	
Empty list	`[]`
Non-empty list	`[Dummy()]`
Empty tuple	`()`
Non-empty tuple	`(Dummy())`
Built-in set and dict types:	
Empty set	`set()`
Non-empty set	`set(Dummy())`
Empty dictionary	`{}`
Non-empty dictionary	`{"a": Dummy()}`
Functions and objects:	
Callable	`Dummy`
Resource	`DummyResource()`
Object	`Dummy()`

Table 2: Coarse-grained abstraction and two modes for concretizing values.

Abstract class of values	Concretization (Python)	
	Deterministic	Randomized
Common primitive values:		
None	`None`	
Boolean	`True`	`True`, `False`
Built-in numeric types:		
Integer	`1`	`-1`, `0`, `1`
Float	`1.0`	`-1.0`, `0.0`, `1.0`
Strings:		
String	`"a"`	`""`, `"a"`
Built-in sequence types:		
List	`[Dummy()]`	`[]`, `[Dummy()]`
Tuple	`(Dummy())`	`()`, `(Dummy())`
Built-in set and dict types:		
Set	`set(Dummy())`	`set()`, `set(Dummy())`
Dictionary	`{"a": Dummy()}`	`{}`, `{"a": Dummy()}`
Functions and objects:		
Callable	`Dummy`	
Resource	`DummyResource()`	
Object	`Dummy()`	

Code Instrumentation

Original code:

```
x = foo()
y = x.bar + z
```

Instrumented code:

```
x = _c_(536, _n_(535, "foo", lambda: foo))
y = _a_(538, _n_(537, "x", lambda: x), "bar") \
        + _n_(539, "z", lambda: z)
```

RQ3: Efficiency at Guiding Executions

- Instrumentation time

  4.5 ms per LoC on average

- Execution time

Table 6: Average execution time (ms) per LoC.

Approach	Dataset	
	Functions	Stack Overflow
CodeT5 FG	178.69	47.29
CodeT5 CG (deterministic)	185.08	46.23
CodeT5 CG (randomized)	167.48	46.31
CodeBERT FG	464.83	133.76
CodeBERT CG (deterministic)	479.89	126.47
CodeBERT CG (randomized)	438.64	127.20
Random	3.94	5.93
Frequency	3.61	5.73
Naive	3.62	5.42
As Is	1.50	5.19

RQ4: Finding Semantics-Changing Commits

- Dataset: 1,000 most recent commits from each project used for evaluation that change a single function.

Table 7: Results from finding semantics-changing commits.

Project	Commits			
	Total	Exceptional	Same behavior	Semantics-changing
Black	68	41	27	0
Flask	114	78	36	0
Pandas	611	403	207	1
Scrapy	522	292	220	10
TensorFlow	320	241	77	2
Total	1,635	1,055	567	13

Average Missing Values

- Open-source functions: 13
- SO snippets: 7