Neuro-Symbolic Developer Tools for Analyzing, Executing, and Repairing Code

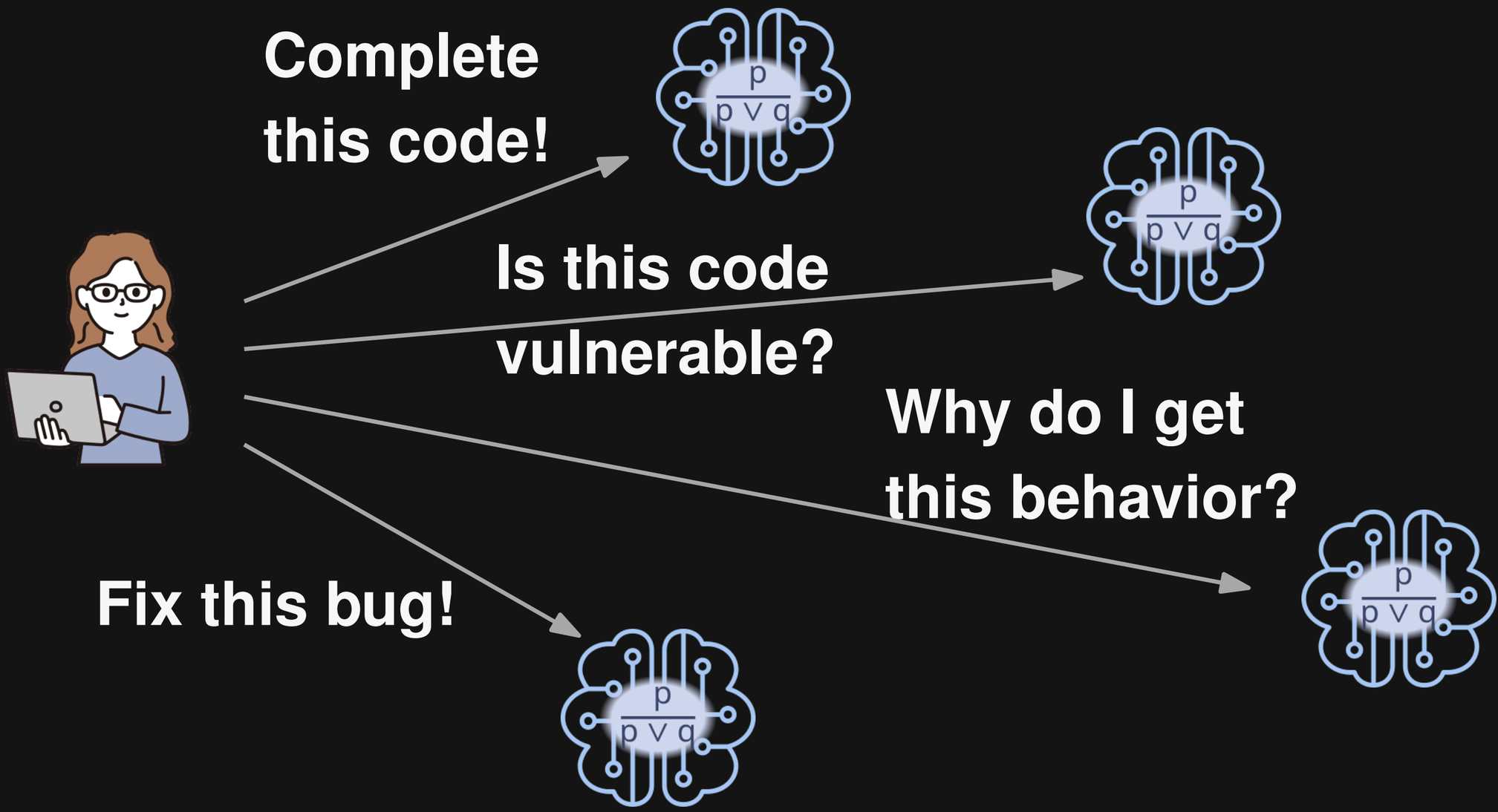
Michael Pradel

University of Stuttgart

Developers Need Tools

Avg. yearly salary:
73k Euro (Germany),
USD 125k (USA)

Developers Need Tools



program analysis

How to create an effective program analysis?

Traditional answer:
Symbolic reasoning

- Manually crafted,
  logic-based rules
- Deterministic, precise reasoning
- Based on formal PL semantics

Traditional answer: Symbolic reasoning

- Manually crafted, logic-based rules
- Deterministic, precise reasoning
- Based on formal PL semantics

Recent answer: Neural reasoning

- Models learned from data (e.g., huge amounts of code)
- Probabilistic reasoning
- Based on “naturalness” of code

Traditional answer: Symbolic reasoning

- Manually crafted, logic-based rules
- Deterministic, precise reasoning
- Based on formal PL semantics

Recent answer: Neural reasoning

- Models learned from data (e.g., huge amounts of code)
- Probabilistic reasoning
- Based on “naturalness” of code

→ Needs heuristics to be practical

→ Fails to understand developer intention

Traditional answer: Symbolic reasoning

- Manually crafted, logic-based rules
- Deterministic, precise reasoning
- Based on formal PL semantics

→ **Needs heuristics to be practical**

→ **Fails to understand developer intention**

Recent answer: Neural reasoning

- Models learned from data (e.g., huge amounts of code)
- Probabilistic reasoning
- Based on “naturalness” of code

→ **Easily misses well-known facts and rules**

→ **Hard to understand and debug**

Traditional answer: Symbolic reasoning

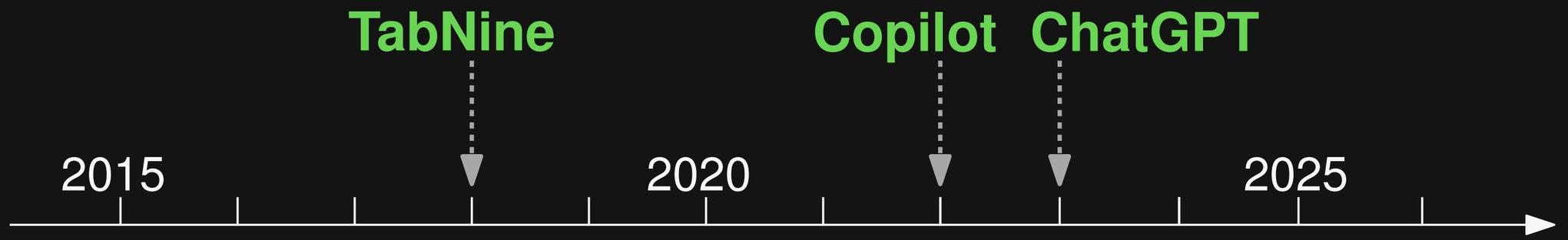
- Manually crafted, logic-based rules
- Deterministic, precise reasoning
- Based on formal PL semantics

Recent answer: Neural reasoning

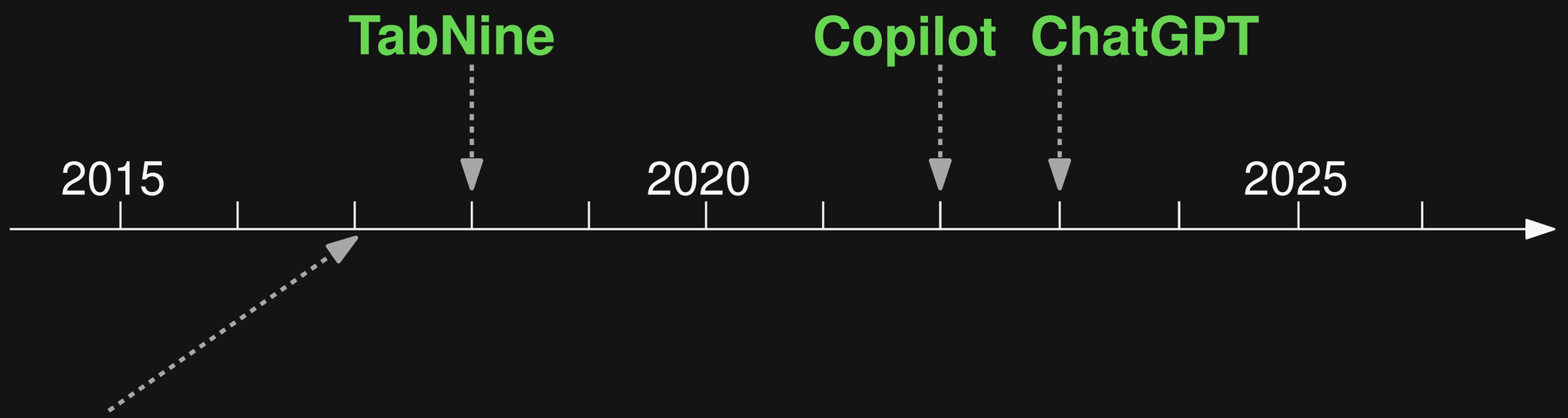
- Models learned from data (e.g., huge amounts of code)
- Probabilistic reasoning
- Based on “naturalness” of code

Get the best of both worlds: Neuro-symbolic developer tools

A Bit of History



A Bit of History



DeepBugs

(OOPSLA'18)

Bug detection as a neural classification problem

Example: DeepBugs

```
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
```

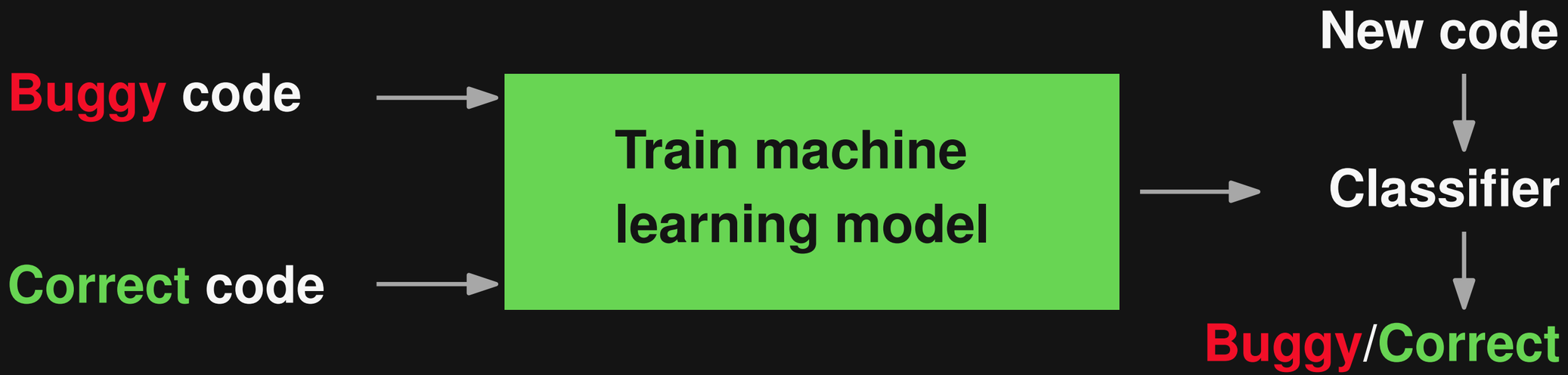
Example: DeepBugs

```
function setPoint(x, y) { ... }

var x_dim = 23;
var y_dim = 5;
setPoint(y_dim, x_dim);
```

Incorrect order of arguments

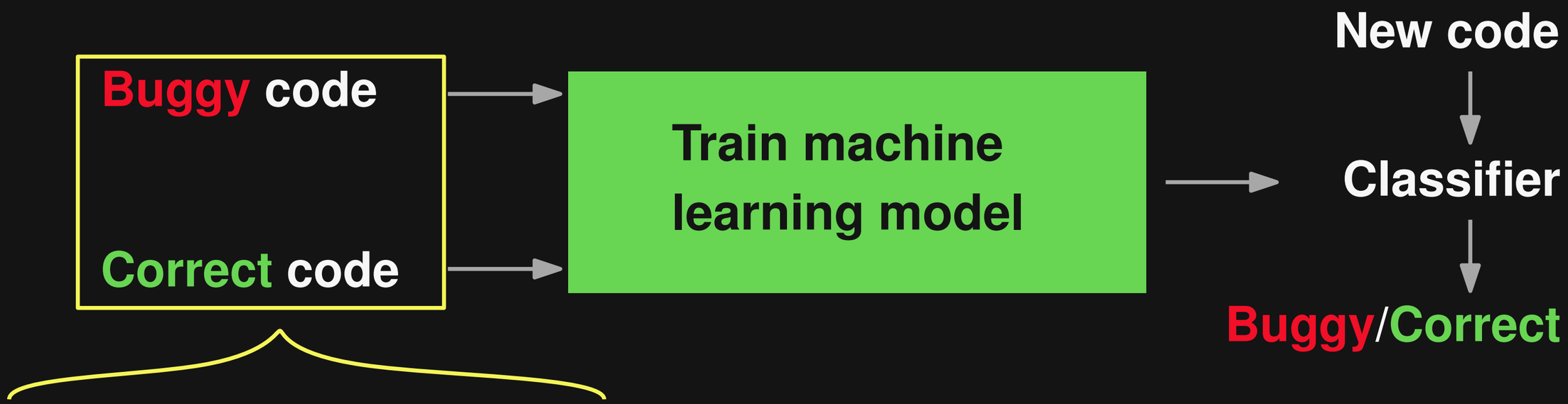
DeepBugs: Learning to Find Bugs

Train a model to distinguish correct from buggy code

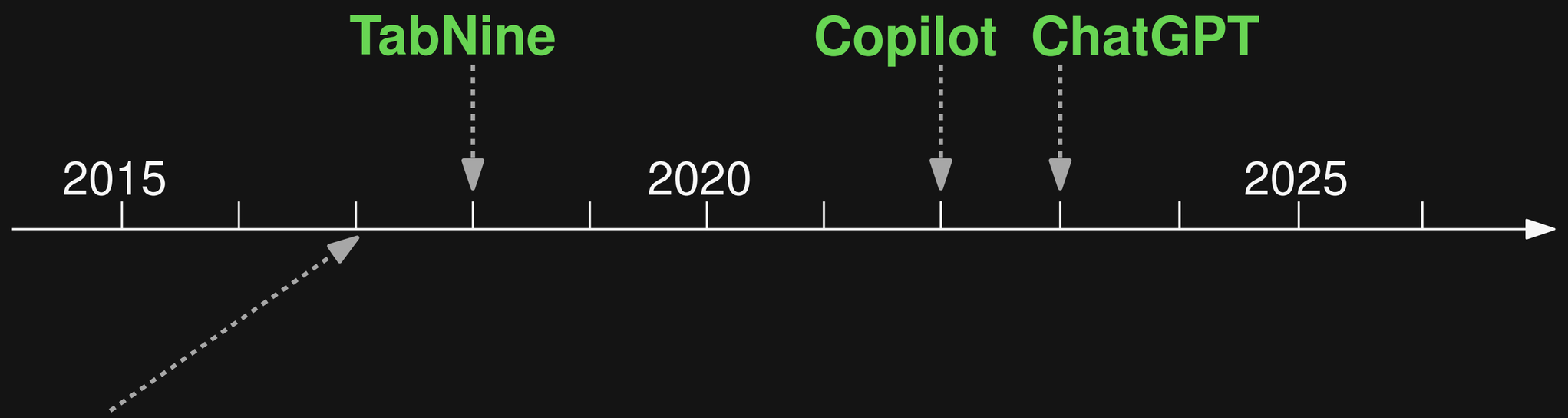


DeepBugs: Learning to Find Bugs

Train a model to distinguish correct from buggy code



A Bit of History

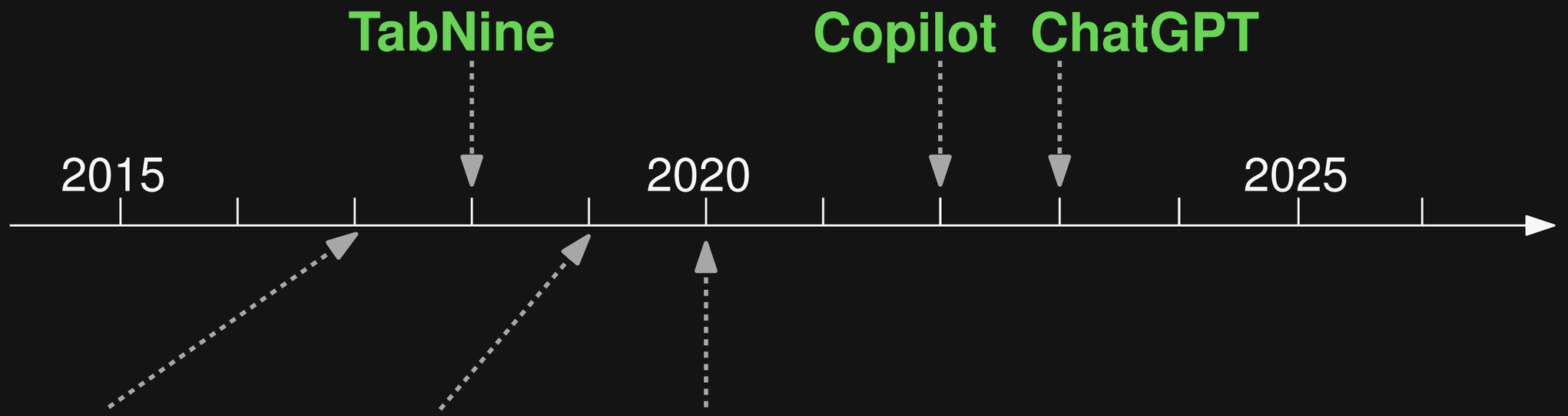


DeepBugs

(OOPSLA'18)

Bug detection as a neural classification problem

A Bit of History



DeepBugs
(OOPSLA'18)

Bug detection as a neural classification problem

NL2Type & **TypeWriter**
(ICSE'19) (FSE'20)

Neural type inference with static validation

Example: TypeWriter

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Example: TypeWriter

```
def find_match(color):
  """
  Args:
    color (str): color to match on and return
  """
  candidates = get_colors()
  for candidate in candidates:
    if color == candidate:
      return color
  return None

def get_colors():
  return ["red", "blue", "green"]
```

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example: TypeWriter

```
def find_match(color):
    """
    Args:
        color (str): color to
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

Top-most predictions:
Type errors

Predictions:
1) int
2) str
3) bool

Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

Example: TypeWriter

```
def find_match(color):
    """
    Args:
        color (str): color to
    """
    candidates = get_colors()
    for candidate in candidates:
        if color == candidate:
            return color
    return None

def get_colors():
    return ["red", "blue", "green"]
```

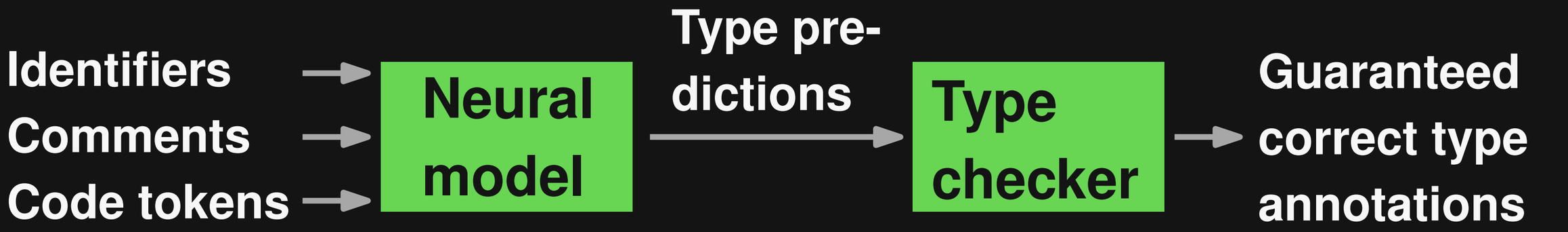
Type-correct predictions

Predictions:
1) int
2) str
3) bool

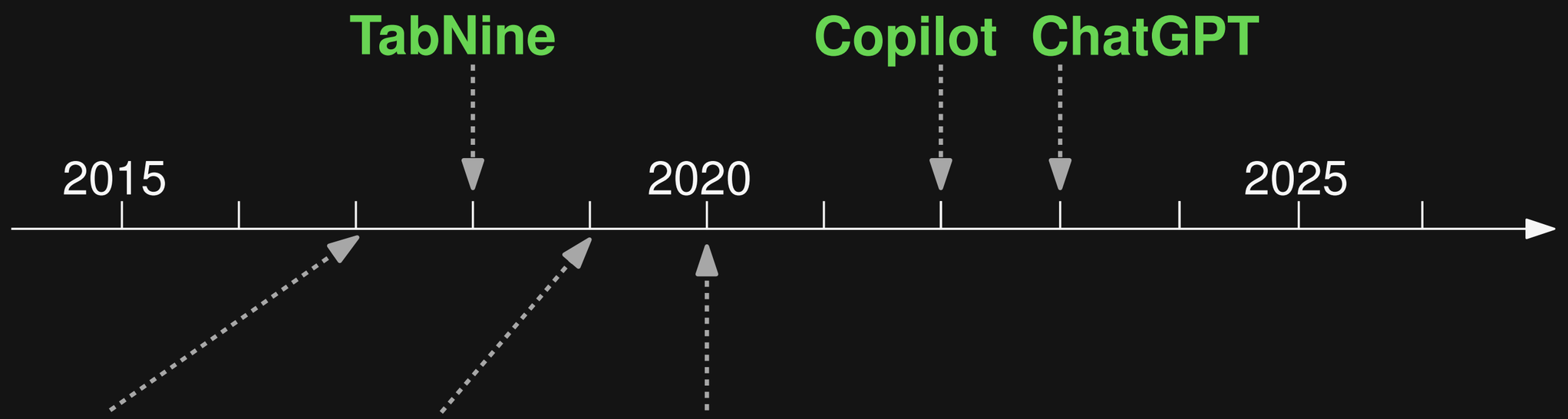
Predictions:
1) str
2) Optional[str]
3) None

Predictions:
1) List[str]
2) List[Any]
3) str

TypeWiter: Neural Type Prediction



This Talk



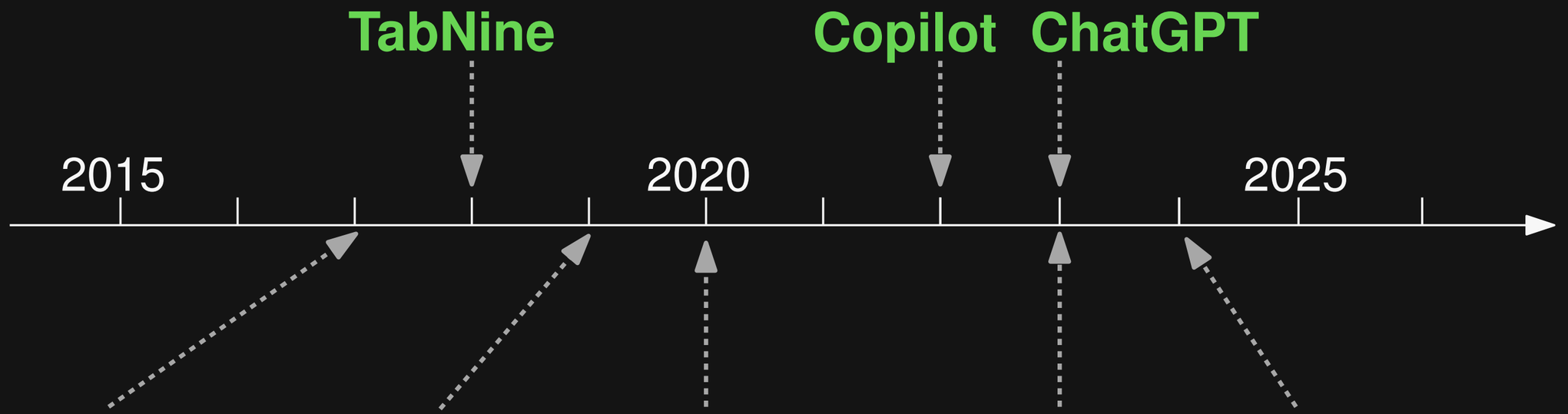
DeepBugs
(OOPSLA'18)

Bug detection as a neural classification problem

NL2Type & **TypeWriter**
(ICSE'19) (FSE'20)

Neural type inference with static validation

This Talk



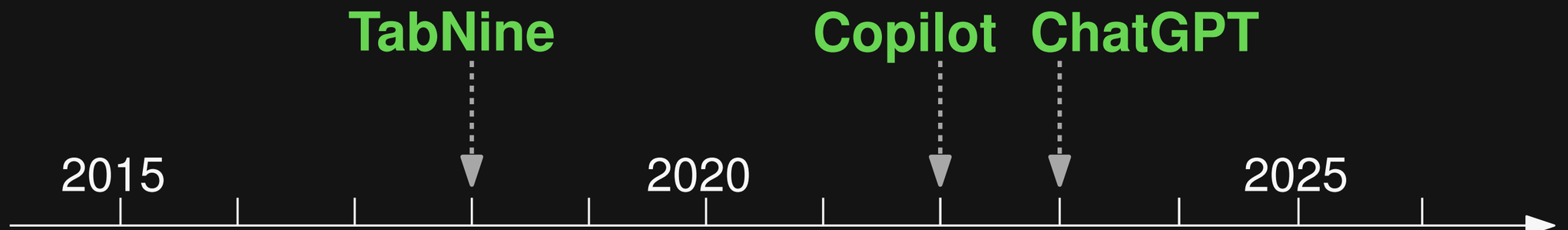
DeepBugs
(OOPSLA'18)
Bug detection as a neural classification problem

NL2Type & **TypeWriter**
(ICSE'19) (FSE'20)
Neural type inference with static validation

LExecutor
(FSE'23)
Learning-guided execution

RepairAgent
(ICSE'25)
Autonomous, LLM-based program repair

This Talk



DeepBugs
(OOPSLA'18)
Bug detection as a neural classification problem

NL2Type & TypeWriter
(ICSE'19) (FSE'20)
Neural type inference with static validation

LExecutor
(FSE'23)
Learning-guided execution

RepairAgent
(ICSE'25)
Autonomous, LLM-based program repair

Focus of this talk

Motivation

Imagine you want to execute this code:

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

train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]

# ...
```

Motivation

Imagine you want to execute this code:


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

train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]

# ...
```

Motivation

Imagine you want to execute this code:


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

train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]

# ...
```

Motivation

Imagine you want to execute this code:

Missing function

Missing variable

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


train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]


# ...
```

Missing variable

Motivation

Imagine you want to execute this code:

Missing function

Missing variable

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


train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]


# ...
```

Missing import and attribute

Missing variable

Why Execute Incomplete Code?

Lots of incomplete code

- Code generated by language models
- Code extracted from deep inside complex projects

Execution enables dynamic analysis, e.g.,

- Check for exceptions and assertion violations
- Compare two code snippets for semantic equivalence

LExecutor

Learning-guided approach for executing arbitrary code snippets

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

Underconstrained execution:
No guarantee that values are realistic

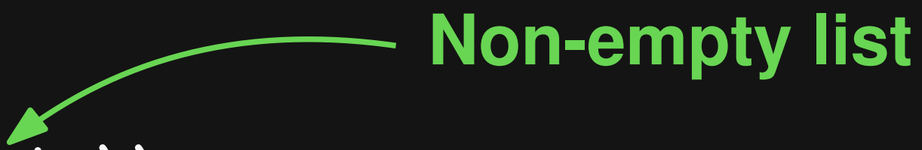
Example: LExecutor

Let’s “lexecute” the motivating example:

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

train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]

# ...
```

Example: LExecutor

Let’s “lexecute” the motivating example:




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

train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]

# ...
```

Example: LExecutor

Let's "lexecute" the motivating example:


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

train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]

# ...
```

Example: LExecutor

Let's "lexecute" the motivating example:

Function that returns `True`

Non-empty list

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

train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]

# ...
```

Non-empty string

Example: LExecutor

Let's "lexecute" the motivating example:

Function that returns `True`

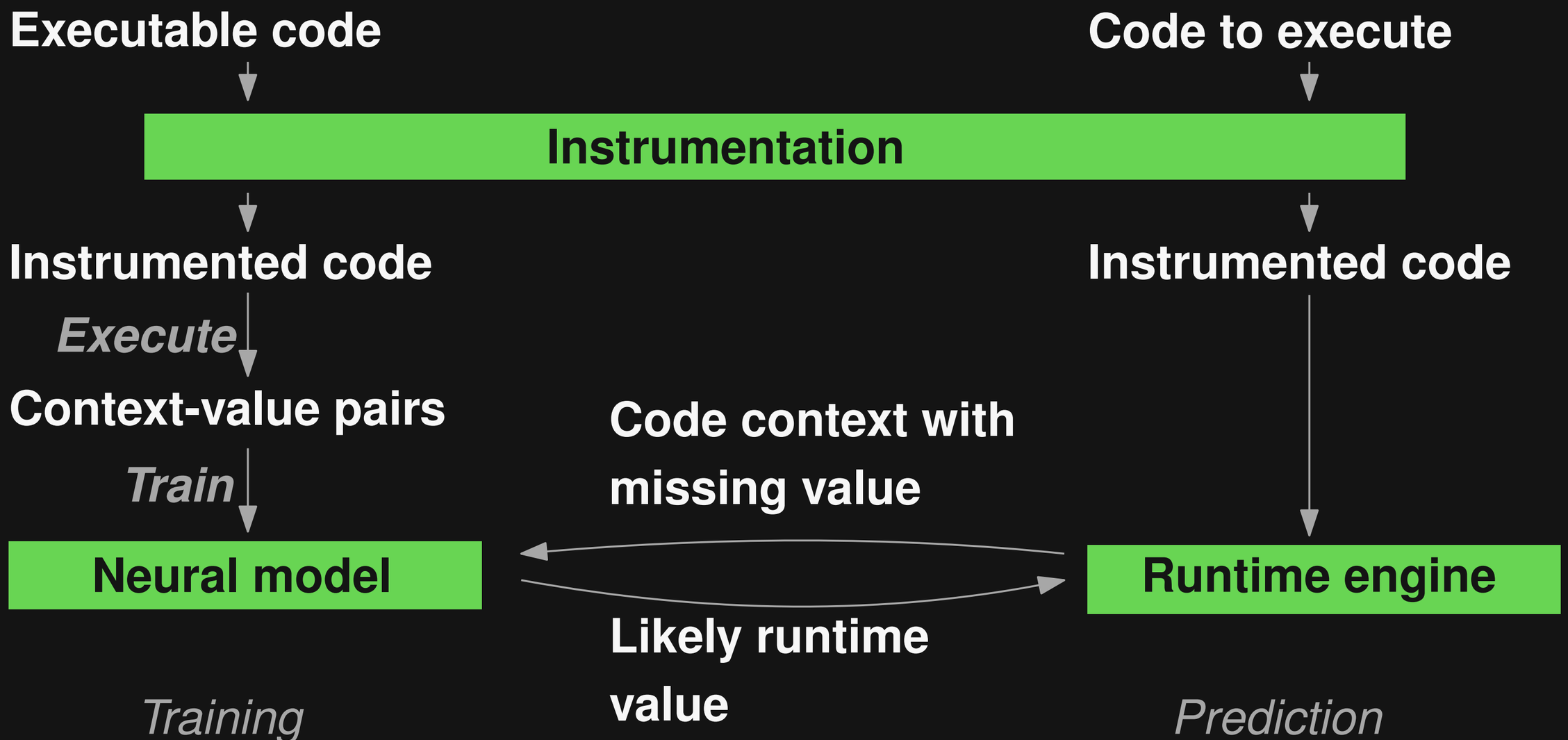
Non-empty list

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


train_len = round(0.8 * len(all_data))
logger.info(f"Extracting data with {config_str}")
train_data = all_data[0:train_len]


# ...
```

Object with a method

Non-empty string

Overview of LExecutor



Example: Code Instrumentation

Original code:

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

Example: 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)
```

Example: 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)
```

Lambda function to postpone the read (to be called by our runtime engine)

Example: Code Instrumentation

Original code:

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

Wrapper for variable reads

Lambda function to postpone the read (to be called by our runtime engine)

Instrumented code:

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

Example: Code Instrumentation

Original code:

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

Wrapper for variable reads

Lambda function to postpone the read (to be called by our runtime engine)

Instrumented code:

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

Wrapper for calls

Example: Code Instrumentation

Original code:

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

Wrapper for variable reads

Lambda function to postpone the read (to be called by our runtime engine)

Instrumented code:

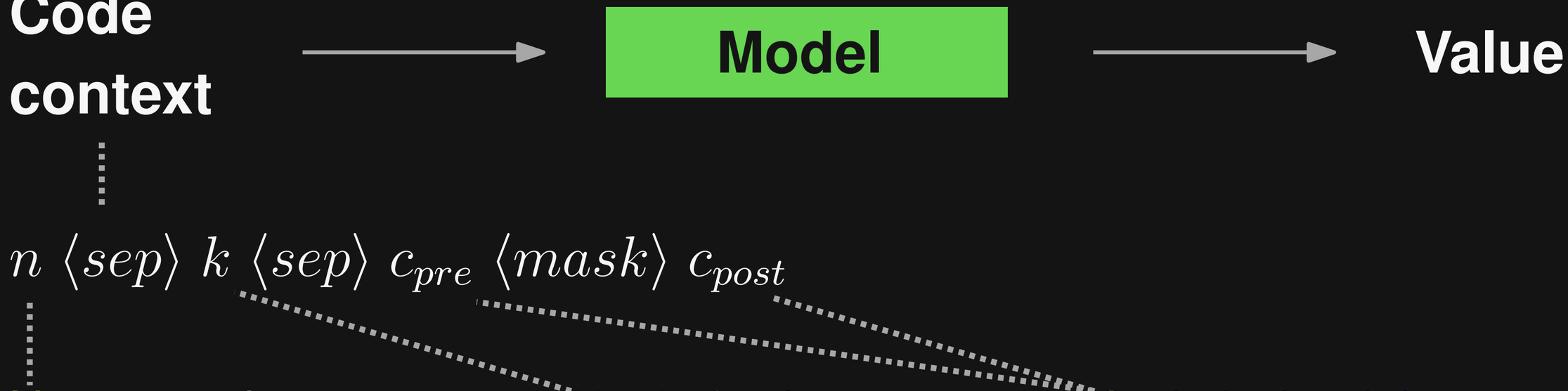
```
x = _c_(536, _n_(535, "foo", lambda: foo))
y = _a_(538, _n_(537, "x", lambda: x), "bar") \
        + _n_(539, "z", lambda: z)
```

Wrapper for calls

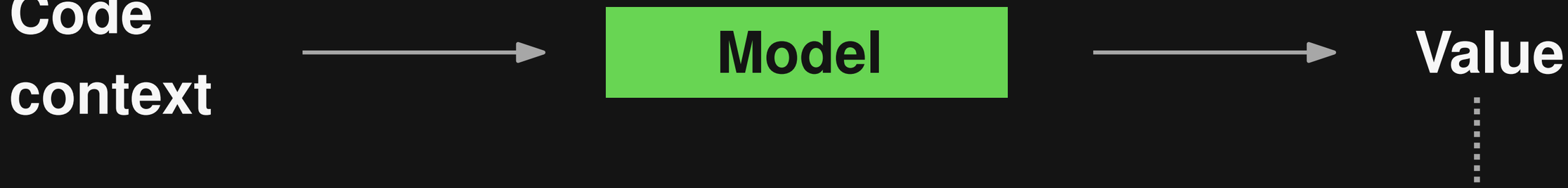
Wrapper for attribute reads

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

Evaluation

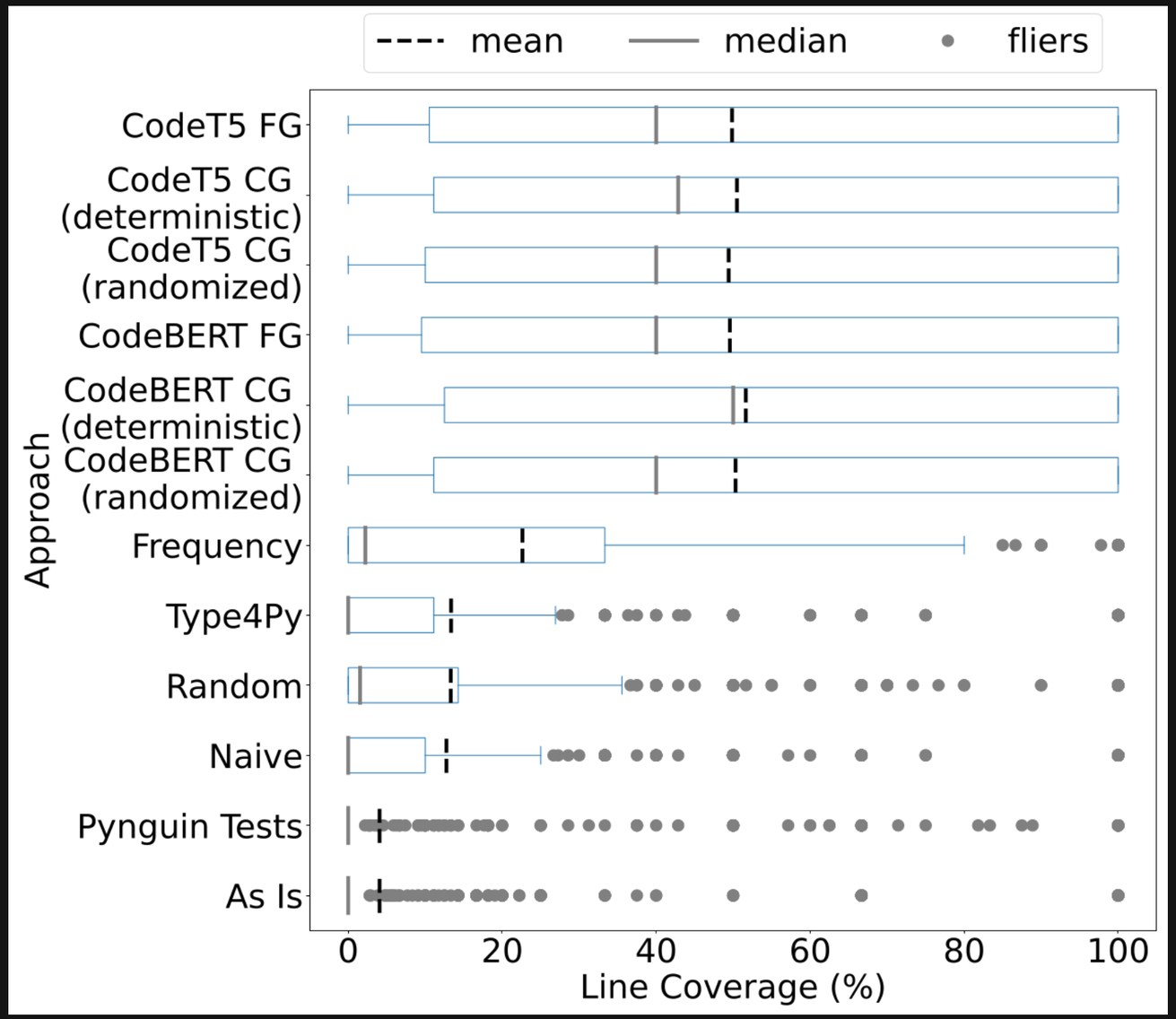
- **Training data**
  - 226k unique value-use events from five projects
- **Code snippets to execute**
  - Open-source functions: 1,000 extracted from five projects
  - Stack Overflow snippets: 462 syntactically correct code snippets in answers to 1,000 Python-related questions

Accuracy

How accurate is the model at predicting realistic values?

	Value abstraction			
	Fine-grained (23 abstract classes of values)		Coarse-grained (12 abstract classes of values)	
	CodeT5	CodeBERT	CodeT5	CodeBERT
Top-1	80.1%	79.5%	88.1%	87.3%
Top-3	88.4%	94.5%	92.1%	96.5%
Top-5	91.7%	96.8%	94.2%	98.2%

Effectiveness at Covering Code (Open-source functions)



Variants of LExecutor

Neural type prediction

State-of-the-art unit test generator

Just run the code

Summary: LExecutor

Symbolic reasoning

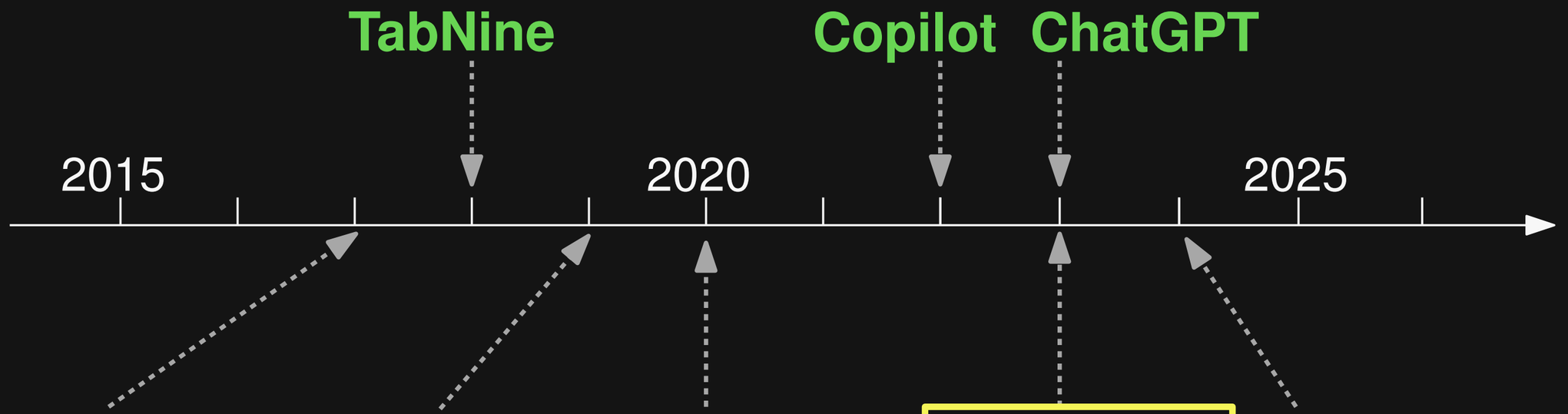
- Execute code using standard PL semantics
- Enables various dynamic analyses

Neural reasoning

- Fill-in missing information on demand during the execution
- Enables execution of otherwise “unexecutable” code

- “*LExecutor: Learning-Guided Execution*” (FSE, 2023, Distinguished Paper Award)
- https://github.com/michaelpradel/LExecutor

Timeline



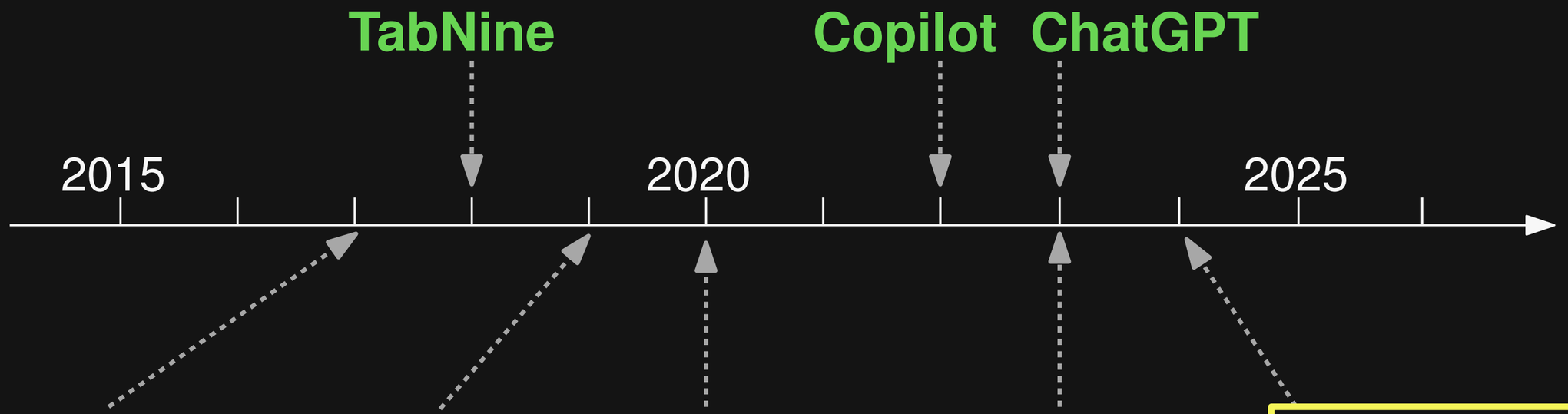
DeepBugs
(OOPSLA'18)
Bug detection as a neural classification problem

NL2Type & **TypeWriter**
(ICSE'19) (FSE'20)
Neural type inference with static validation

LExecutor
(FSE'23)
Learning-guided execution

RepairAgent
(ICSE'25)
Autonomous, LLM-based program repair

Timeline



DeepBugs
(OOPSLA'18)
Bug detection as a neural classification problem

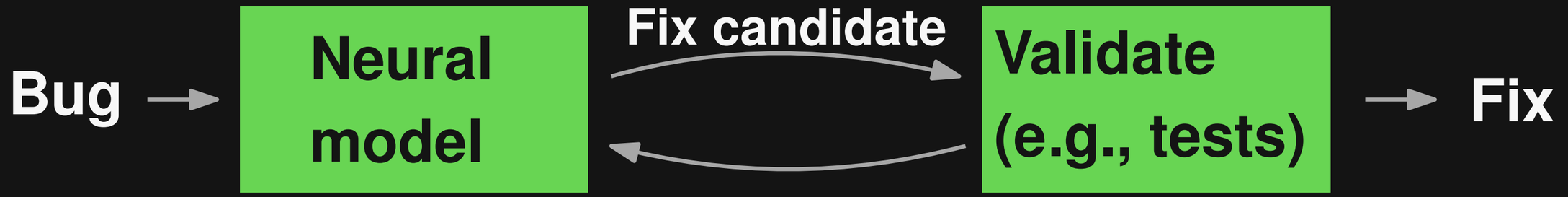
NL2Type & **TypeWriter**
(ICSE'19) (FSE'20)
Neural type inference with static validation

LExecutor
(FSE'23)
Learning-guided execution

RepairAgent
(ICSE'25)
Autonomous, LLM-based program repair

Automated Program Repair

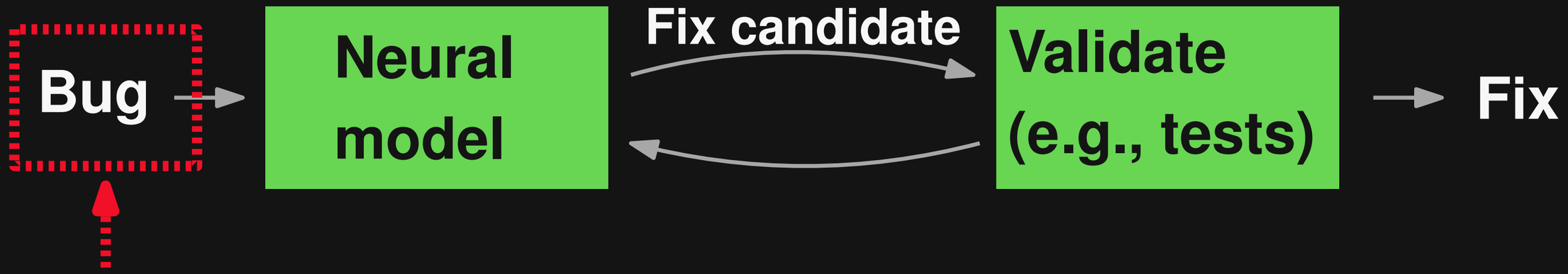
- State of the art: **Learning-based repair**



Automated Program Repair
Claire Le Goues, Michael Pradel, Abhik Roychoudhury
Communications of the ACM, 62(12), 2019

Automated Program Repair

- State of the art: **Learning-based repair**



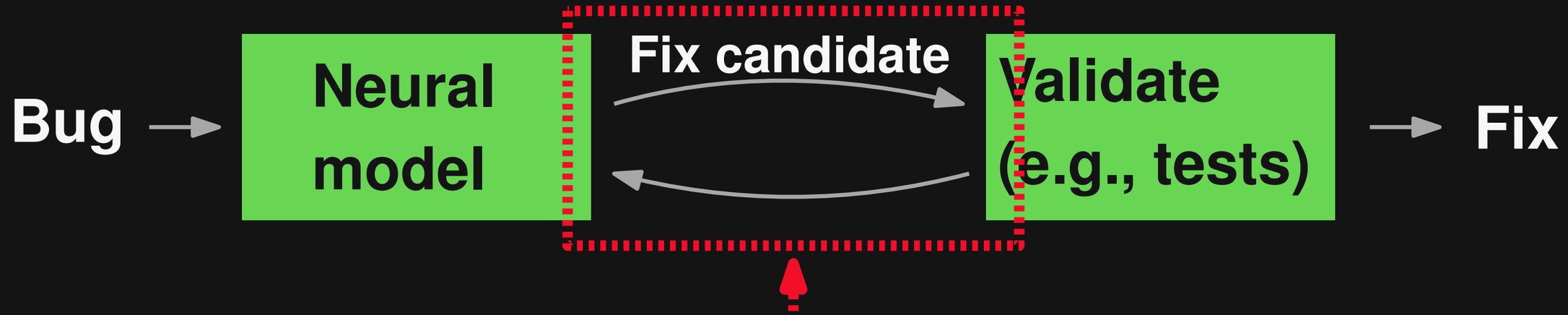
Fixed input: Lines around the bug location

But: Human developers **actively gather additional information**

Automated Program Repair
Claire Le Goues, Michael Pradel, Abhik Roychoudhury
Communications of the ACM, 62(12), 2019

Automated Program Repair

- State of the art: **Learning-based repair**



Fixed algorithm: Generate-and-validate

But: Human developers interact with **various tools** in non-trivial ways

Automated Program Repair
Claire Le Goues, Michael Pradel, Abhik Roychoudhury
Communications of the ACM, 62(12), 2019

RepairAgent

New paradigm: LLM takes the decisions autonomously

- Agent invokes developer tools:
  LLM mimics human-like actions
- Interaction via natural language:
  LLM mimics human-like reasoning

Example: RepairAgent

LLM agent

You are an AI assistant specialized in fixing bugs in Java code. Your objective is to autonomously understand and fix the following bug:
[Details of failing test case]

Use the following tools:
[Details on available tools]

Tools

Example: RepairAgent

LLM agent

Based on the stack trace, I should analyze the code in “ControlFlowAnalysis.java”.

Let’s call the ”read code” tool!

Tools

Example: RepairAgent

LLM agent

```
...
if (cfa != null) {
  for (Node finallyNode:
        cfa.finallyMap.get(parent)) {
    cfa.createEdge(fromNode,
                   Branch.UNCOND,
                   finallyNode);
  }
}
...
```

Tools

Example: RepairAgent

LLM agent

I need to see the failing test case “testDeepNestedFinally” to understand the assertions and inputs that lead to the failure.

Let’s call the “extract test code” tool!

Tools

Example: RepairAgent

LLM agent

```
...
String src = ...;
ControlFlowGraph<Node> cfg = createCfg(src);
assertCrossEdge(cfg, Token.THROW,
                     Token.BLOCK, Branch.ON_EX);
assertCrossEdge(cfg, Token.VAR,
                     Token.BLOCK, Branch.UNCOND);
...
```

Tools

Example: RepairAgent

LLM agent

Based on the test case and the code, I hypothesize that the use of ’Branch.UNCOND’ in ’finally’ blocks is incorrect

Let’s work under this hypothesis!

Tools

Example: RepairAgent

LLM agent

I should look at calls similar to the potentially buggy call to “createEdge”. This will help me understand ...

Let’s use the “extract similar calls” tool!

Tools

Example: RepairAgent

LLM agent

Search results:

```
createEdge(node,
                Branch.ON_TRUE,
                computeFallThrough(thenBlock));

createEdge(cfgNode,
                Branch.ON_EX,
                catchBlock);

...
```

Tools

Example: RepairAgent

LLM agent

Another call uses ’Branch.ON_EX’, which ...

Let’s apply and test this fix!

```
   cfa.createEdge(fromNode,
-                 Branch.UNCOND,
+                 Branch.ON_EX,
                  finallyNode);
```

Tools

Example: RepairAgent

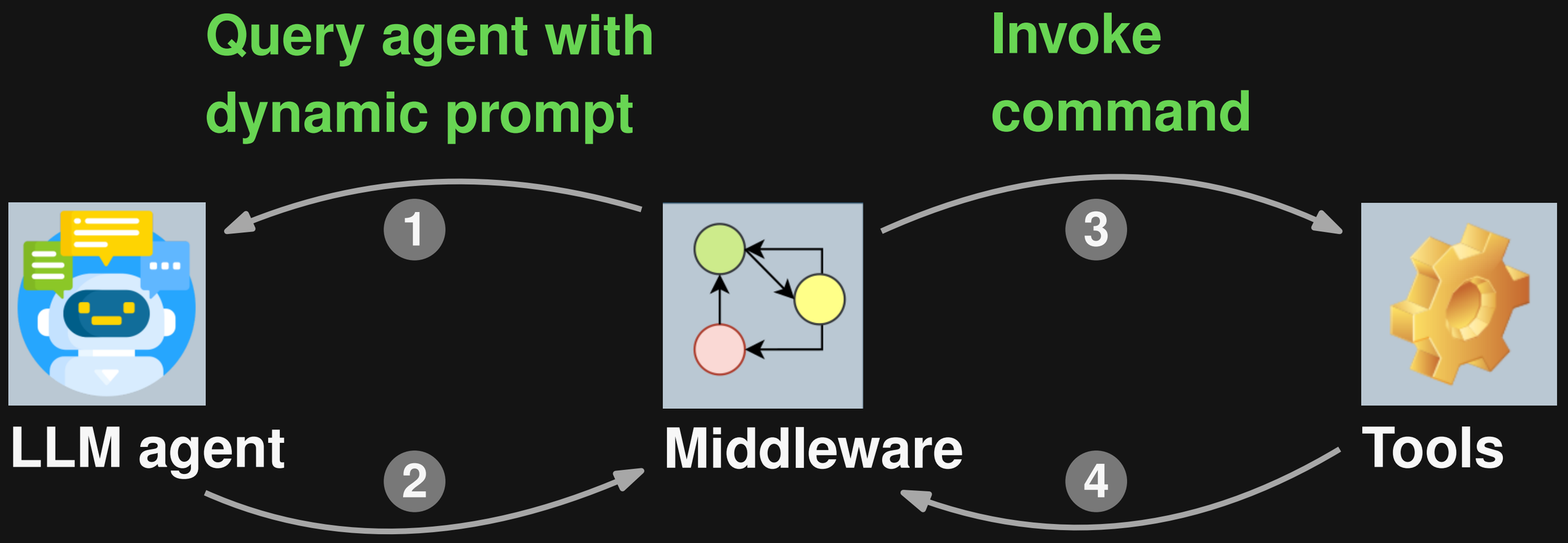
LLM agent

With this fix, all tests pass.

We are done!

Tools

Overview of RepairAgent

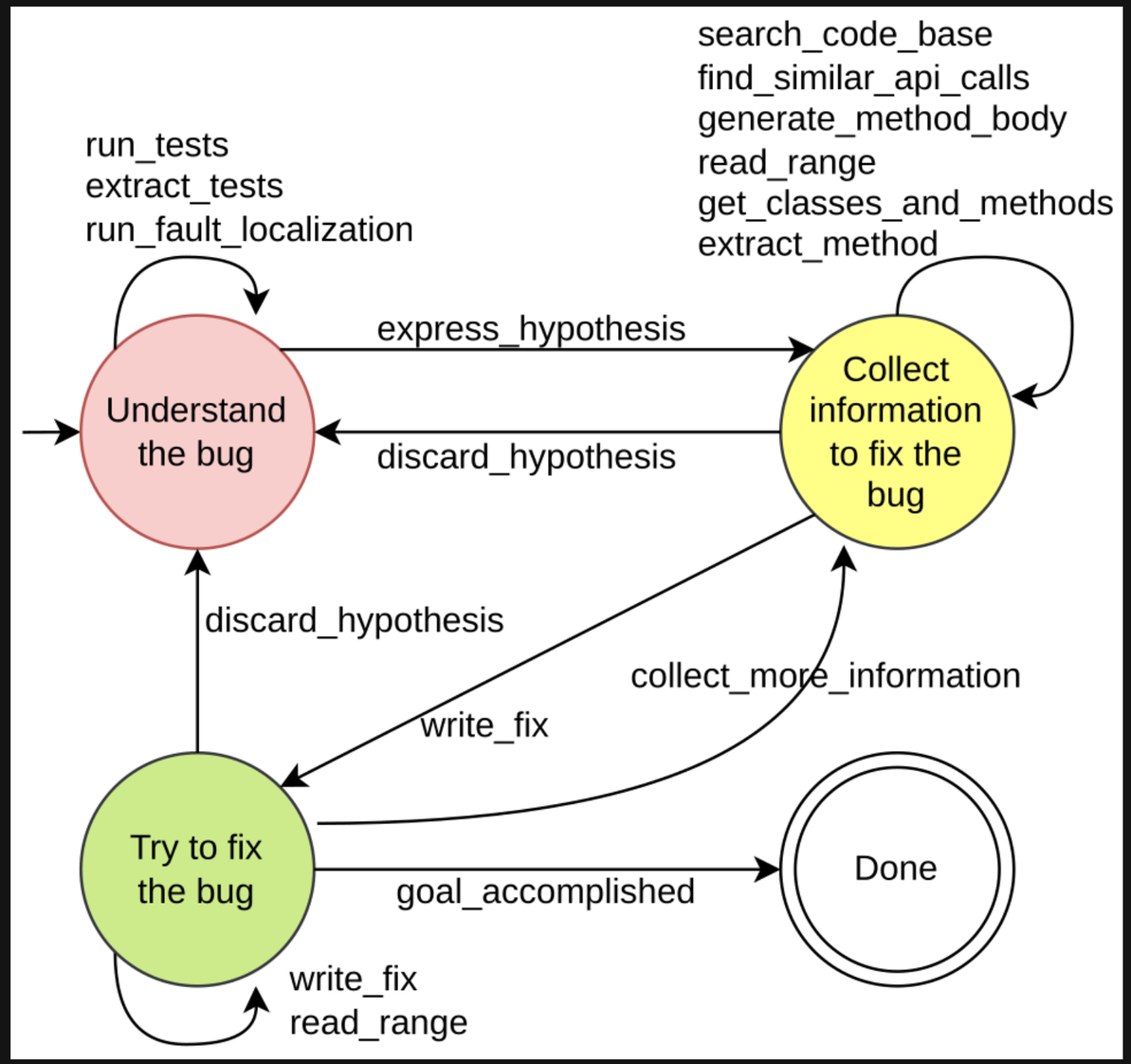


Tools

Category	Tools
Read and extract code	Read range of lines
	Get classes and methods
Search and generate code	Search code base by keyword
	Predict method body via LLM
Testing and patching	Run test suite
	Apply a patch
Control	Express or discard hypothesis
	Declare success

See paper for complete list of 14 tools.

Guidance via Finite State Machine



Evaluation

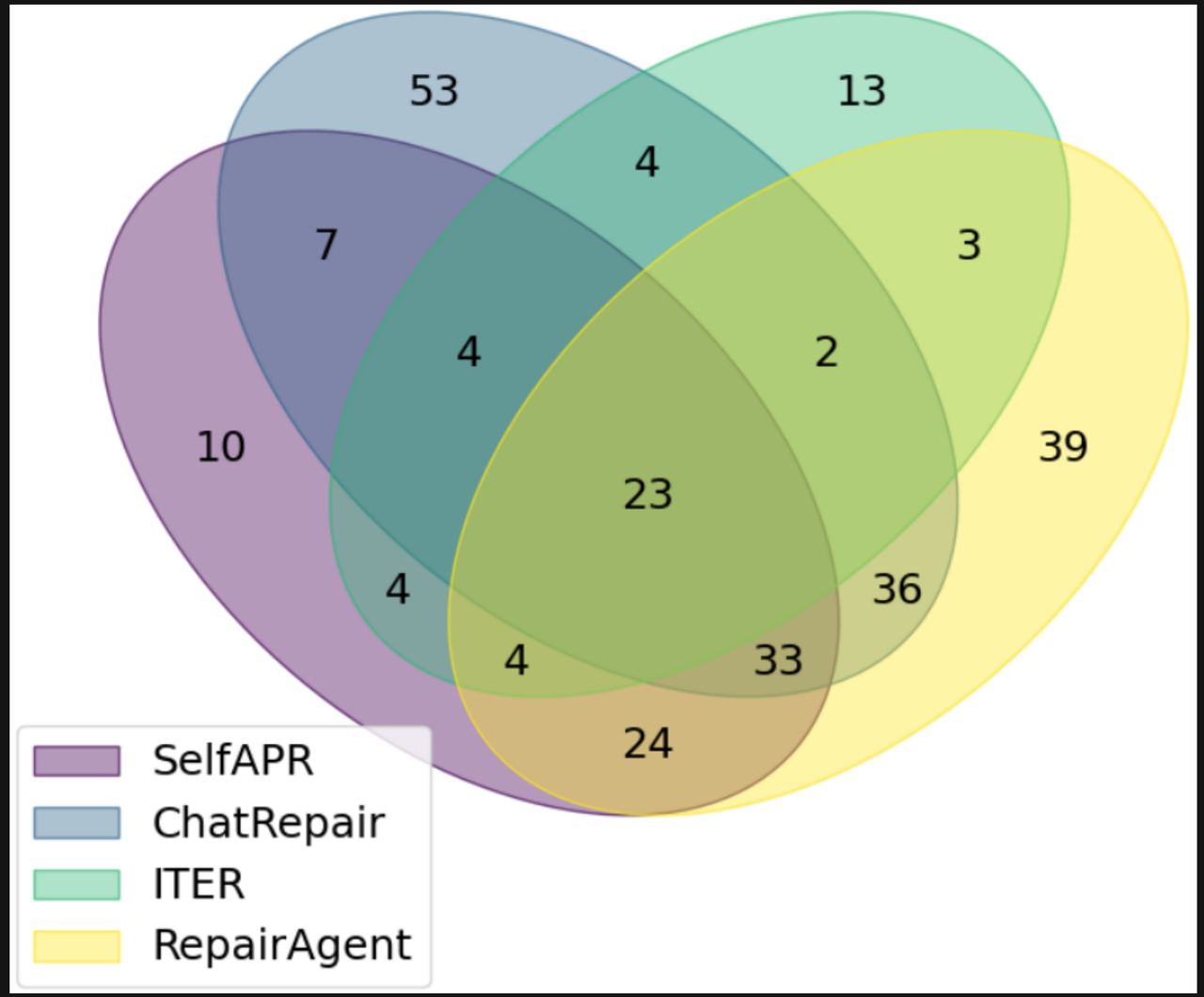
- **OpenAI's GPT-3.5-0125**
- **All 835 bugs from Defects4J v1.2 and v2**
  - Including multi-line, multi-file bugs
- **Measures of success**
  - Plausible fixes
  - Correct fixes
  - Cost per bug

Effectiveness

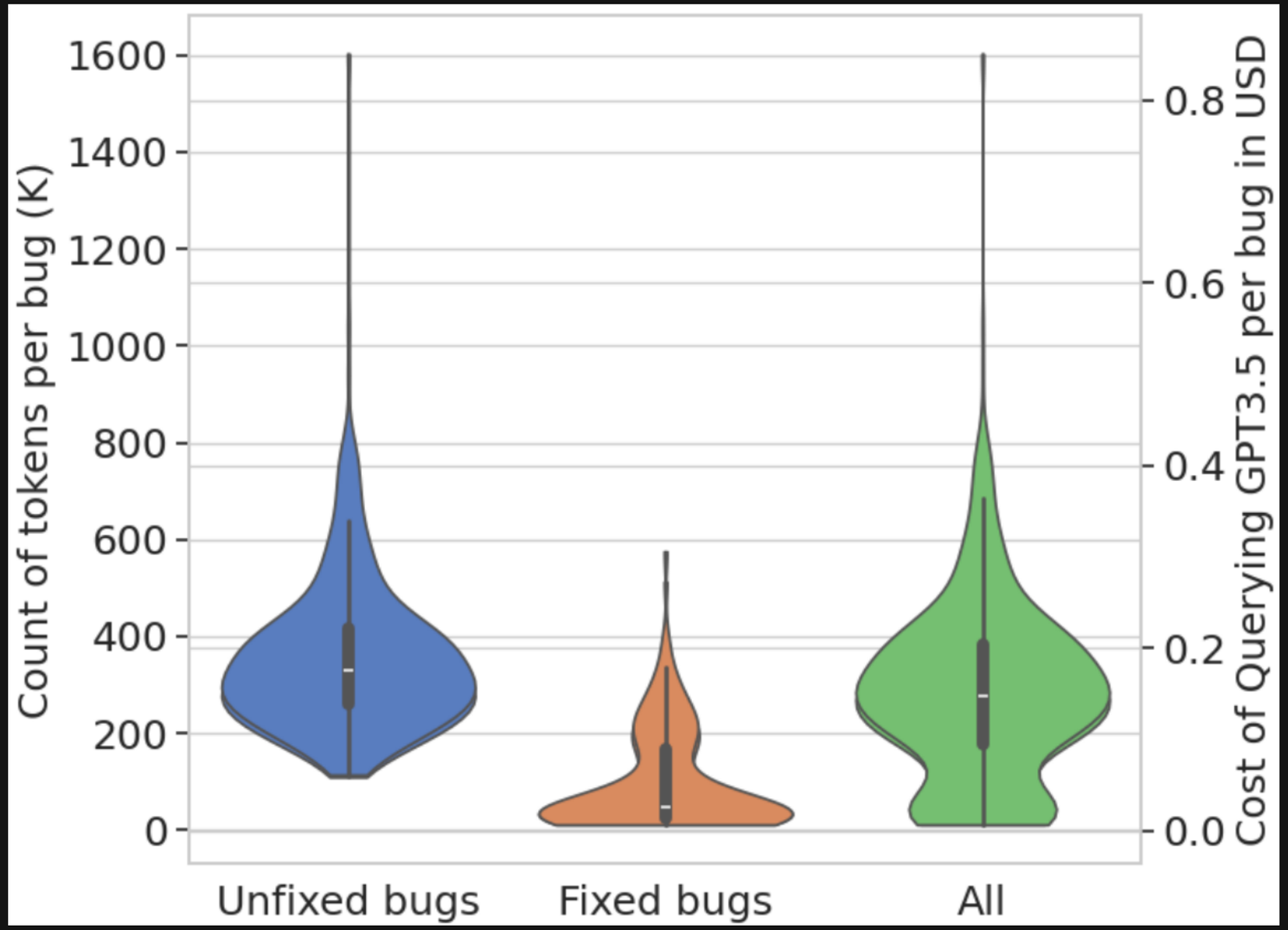
Correct bug fixes:

Bug type	RepairAgent	ChatRepair	ITER	SelfAPR
Single-line	110	133	36	83
Multi-line	46	29	14	24
Multi-file	3	0	4	3
Total	164	162	57	110

Effectiveness



Costs



Avg. per bug:
270k tokens,
USD 0.14

Summary: RepairAgent

Symbolic reasoning

- Test executions
- FSM-based guidance
- Static code search

Neural reasoning

- LLM-driven decision making
- LLM-based code completion
- NL as “glue language”

- “*RepairAgent: An Autonomous, LLM-Based Agent for Program Repair*” (ICSE 2025)
- https://github.com/sola-st/RepairAgent

Conclusions

Neuro-symbolic developer tools are here to stay

- **LExecutor**: Learning-guided execution
- **RepairAgent**: Autonomous, LLM-based repair

Other interests

- LLM-based, universal **fuzzing**
- Reliable **quantum computing**
- Program analysis and benchmarks for **WebAssembly**