Machine Learning for Programming (ML4P)

Course page

http://software-lab.org/teaching/winter2023/ml4p/

Prof. Dr. Michael Pradel

Winter 2023/24

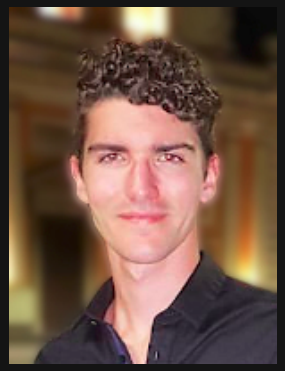
Software Lab, University of Stuttgart

About Me: Michael Pradel

- **Since 9/2019: Full Professor at University of Stuttgart**
- **Before Stuttgart**
  - Studies at TU Dresden, ECP (Paris), and EPFL (Lausanne)
  - PhD at ETH Zurich, Switzerland
  - Postdoctoral researcher at UC Berkeley, USA
  - Assistant Professor at TU Darmstadt
  - Sabbatical at Facebook, Menlo Park, USA

About the Software Lab






- **My research group since 2014**
- **Focus: Tools and techniques for building reliable, efficient, and secure software**
  - Program testing and analysis
  - Machine learning, security
- **Thesis and job opportunities**

Plan for Today

1. Organization

2. Topic of this seminar

Why Have a Seminar?

- **Learn fundamentals of doing research**
  - Read and digest papers
  - Present complex ideas to others
  - Scientific writing
- **Learn about machine learning and program analysis**
  - Exciting and “hot” research area with highly relevant practical applications
  - Maybe your future thesis topic

Organization

- **Today: Kick-off meeting**
- **During the semester**
  - Meetings with mentor
  - Talks by students
- **Your tasks:**
  - Term paper
  - Talk
  - Active participation

Organization

- **Today: Kick-off meeting**
- **During the semester**
  - Meetings with mentor
  - Talks by students
- **Your tasks:**
  - Term paper
  - Talk
  - Active participation

Grading:

40%

40%

20%

Talk

- **20 minutes + questions**
- **English**
- **Present a recent research paper and how it compares to closely related work**
- **Your mentor will help you prepare the presentation**
  - Ask questions about the paper
  - Send slides one week before the talk
  - Incorporate feedback given by the mentor

Talk: Some Advice

Content:

- No need to explain all technical details
- But: Must contain some ”meat”

Presentation:

- Examples are your secret weapon
- Stick to the time limit
- Practice, practice, practice

Pro tip: View video *How to give a good research talk* by Simon Peyton Jones

Talk: Rules

- **Prepare your own slides**
  - No copy & paste from existing slides, even if available
- **You may use examples from the paper**
  - Using your own examples is encouraged

Term Paper

- 6 pages
- English
- LaTeX template on course web site
- Summarize the paper in your own words and discuss it in the context of closely related work
- Must be self-containing

Term Paper: Some Advice

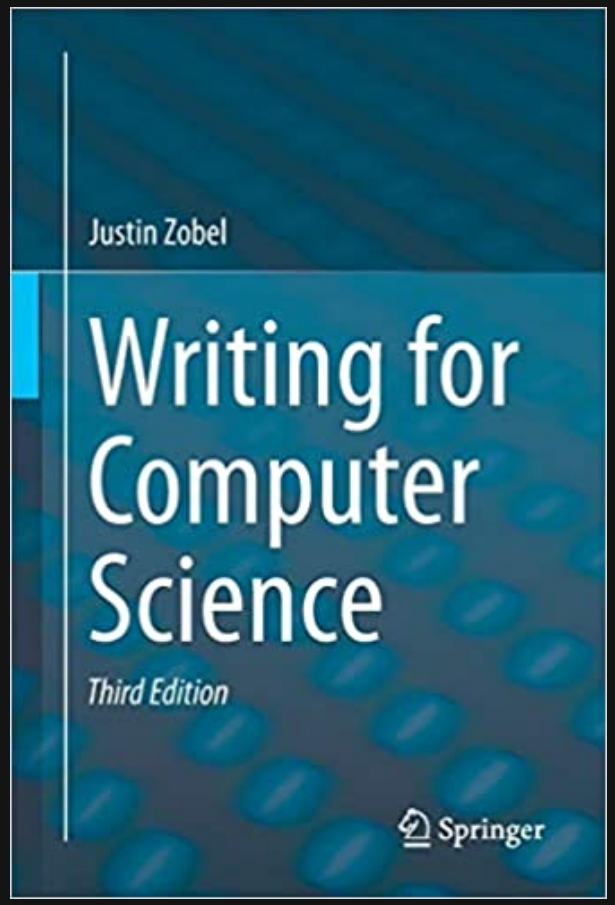
- **Don't waste space on basics**
- **Examples are your secret weapon (yes, again)**
- **Use a neutral perspective**
  - "the analysis" or "the authors", not "we"
- **Bad English distracts from good content**
- **Revise, revise, revise**

General Writing Advice

Great book with many useful tips:

"Writing for Computer Science"

by Justin Zobel



Term Paper: Rules

- **No verbatim copying or paraphrasing of existing text**
  - Exception: Clearly marked, short quotes
- **You may copy figures (e.g., result graphs)**
- **You must use exclusively your own example(s)**

Dates

- **Oct 26, 2023**: Deadline for choosing topics
- **From Nov 9, 2023**: Talks
- **Jan 12, 2024**:
  Draft of term paper
- **Feb 9, 2024**:
  Final term paper

Meetings

- **All meetings are**
  - in the classroom
  - without recording
- **Participation is not mandatory**
  - But: Active participation contributes to the grade

Registering for the “Exam”

- **As with all other courses: Students must register for the exam**
  - Prerequisite for obtaining a grade
- **“Exam” here means participating in the course**
  - No written exam at end of semester

Topics To Choose From

- **Recently published research papers:**
  http://software-lab.org/teaching/winter2023/ml4p/
- **Submit your preferences until next Thursday (Oct 26, end of day)**
  - You pick three topics, we assign one
  - Indicate your preferences in a mail to katharina.plett@iste.uni-stuttgart.de

Plan for Today

1. Organization ✓

2. Topic of this seminar

Topic of This Seminar

Machine Learning for Programming

Topic of This Seminar

Machine Learning for Programming

- Tools for improving software reliability and security
- E.g., automated bug detection, code completion, and program repair

Topic of This Seminar

Machine Learning for Programming

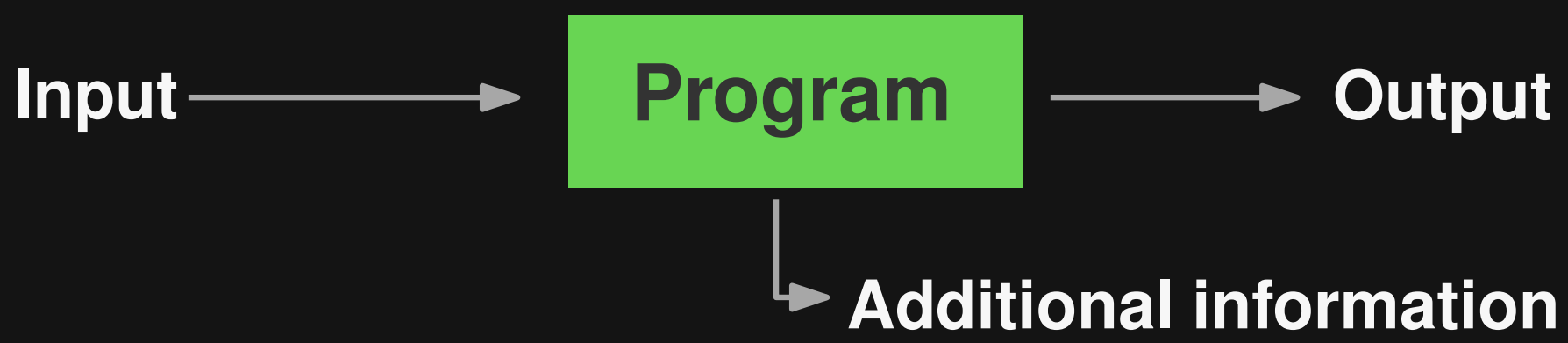
- Source code as data
- Large code corpora to learn from
- Train models that predict program properties

What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities

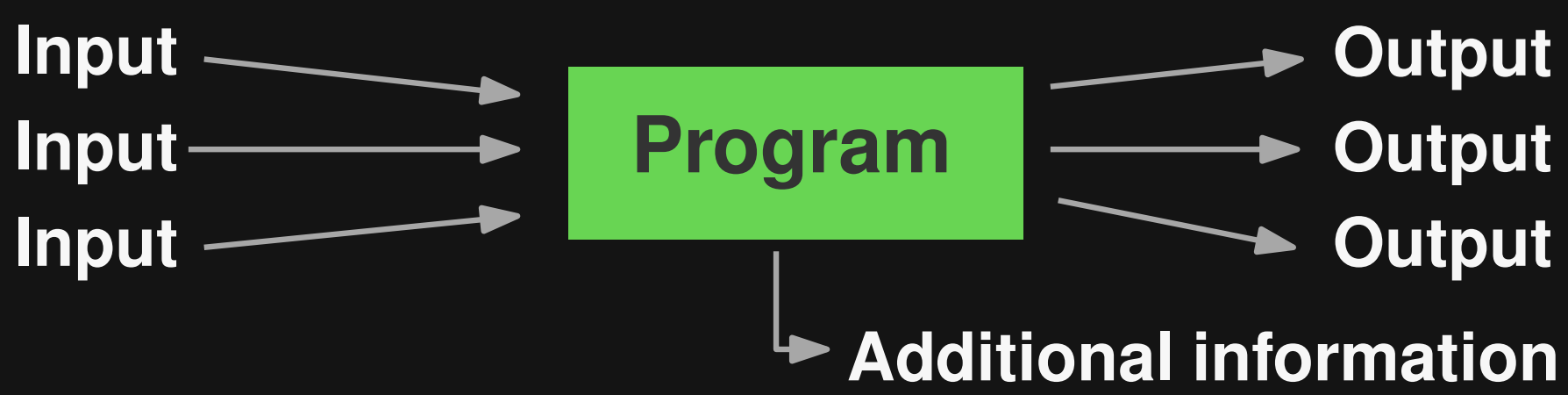
What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities



What is Program Analysis?

- **Automated analysis of program behavior, e.g., to**
  - find programming errors
  - optimize performance
  - find security vulnerabilities



Why Do We Need It?

Basis for various tools that make developers productive

- Compilers
- Bug finding tools
- Performance profilers
- Code completion
- Automated testing
- Code summarization/documentation

Traditional Approaches

- **Analysis has built-in knowledge about the problem to solve**
- **Significant human effort to create a program analysis**
  - Conceptual challenges
  - Implementation effort
- **Analyze a single program at a time**

Neural Software Analysis

Insight: Lots of data about software development to learn from

Source code
Execution traces
Documentation
Bug reports
etc.

→ **Machine Learning** → **Predictive tool**

Neural Software Analysis, Pradel & Chandra, CACM'22

Neural Software Analysis

Insight: Lots of data about software development to learn from

Source code
Execution traces
Documentation
Bug reports
etc.

→ **Machine Learning** →

New code, execution, etc.
↓
Predictive tool
↓
Information useful for developers

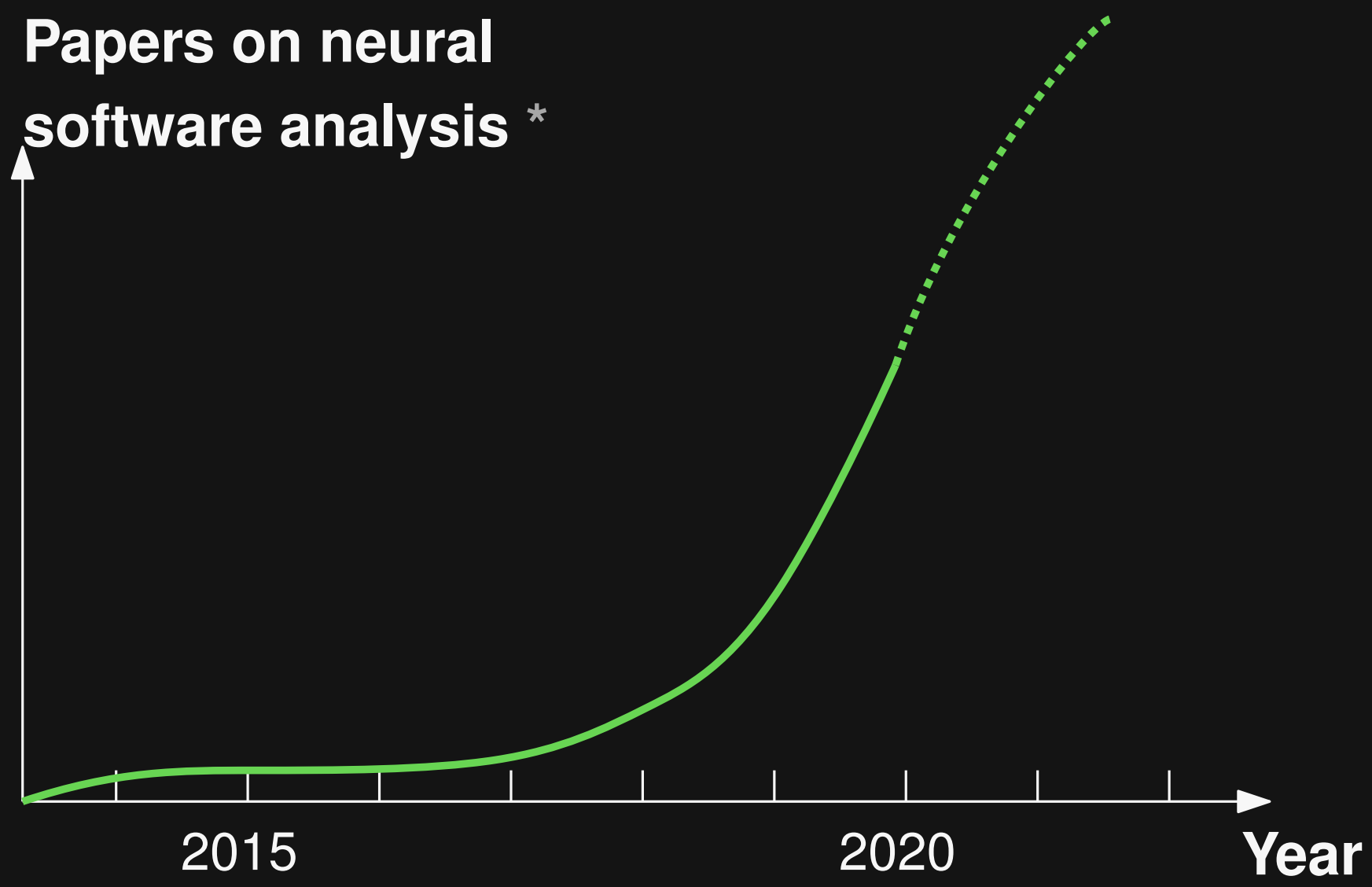
Neural Software Analysis, Pradel & Chandra, CACM'22

Traditional program analysis

- Manually crafted
- Years of work
- Precise, logical reasoning
- Heuristics to handle undecidability
- Challenged by large code bases

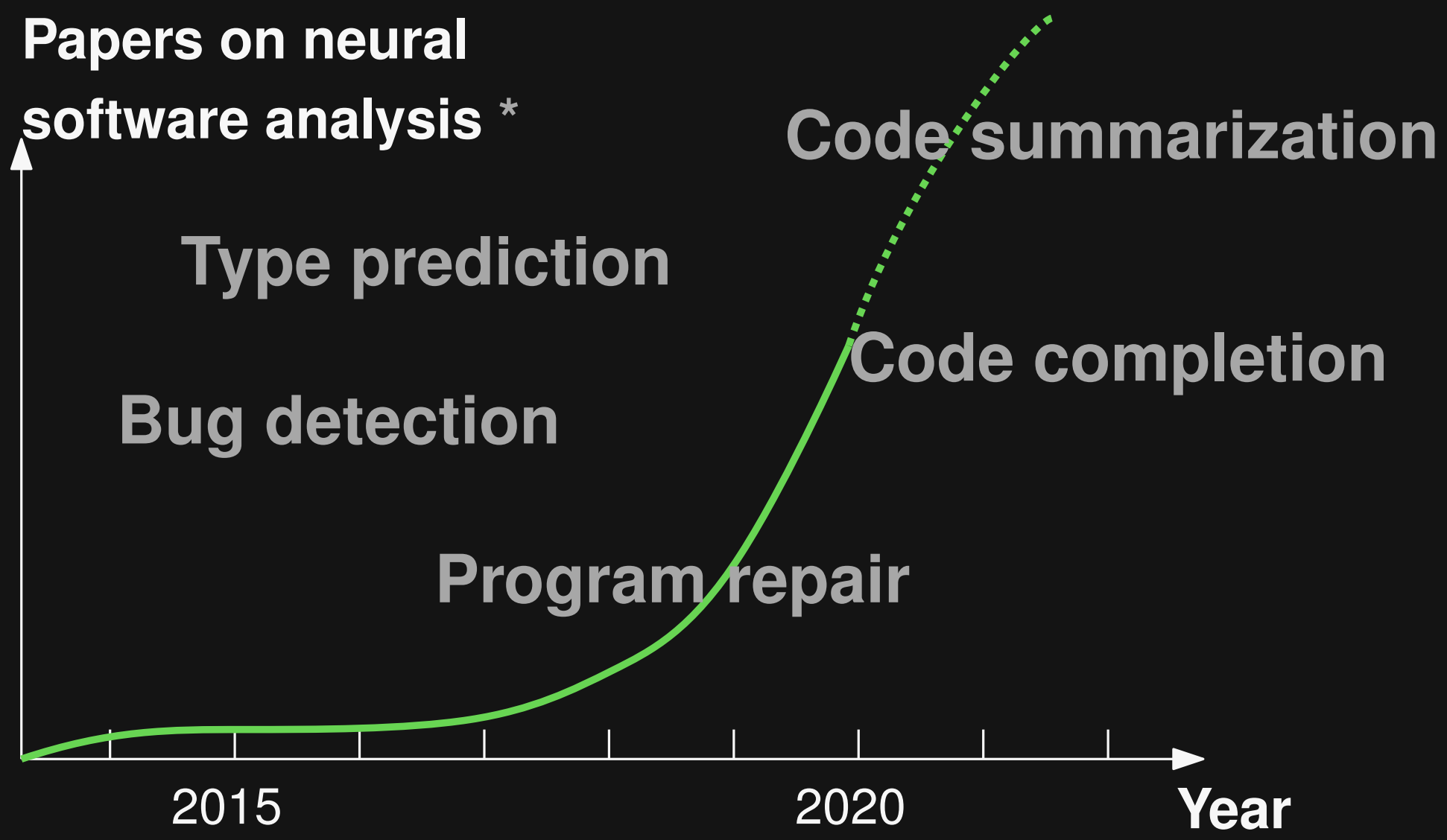
Traditional program analysis		Neural software analysis
Manually crafted	→	Automatically learned within hours or days
Years of work		
Precise, logical reasoning	→	Data-driven prediction
Heuristics to handle undecidability	→	Learn instead of hard-code heuristics
Challenged by large code bases	→	Use big code to our benefit

Join the Hype!



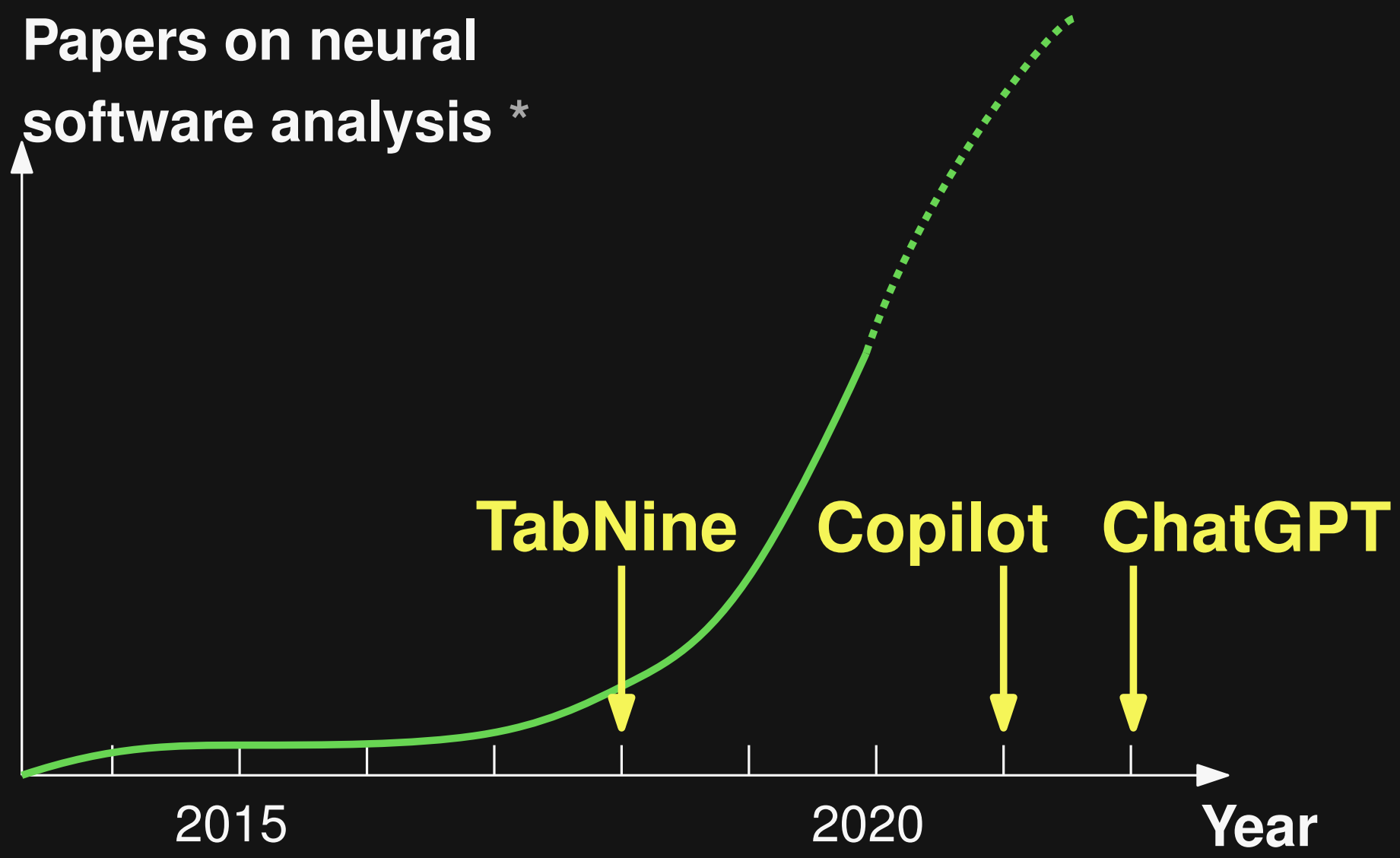
* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

Join the Hype!



* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

Join the Hype!



* Estimate based on *Neural Software Analysis*, Pradel & Chandra, CACM'22

Three Examples

- **Fixing type errors with PyTy**

  PyTy: Repairing Static Type Errors in Python, under review

- **Neural bug detection with CMI-Finder**

  When to Say What: Learning to Find Condition-Message Inconsistencies, ICSE'23

- **Enabling execution with LExecutor**

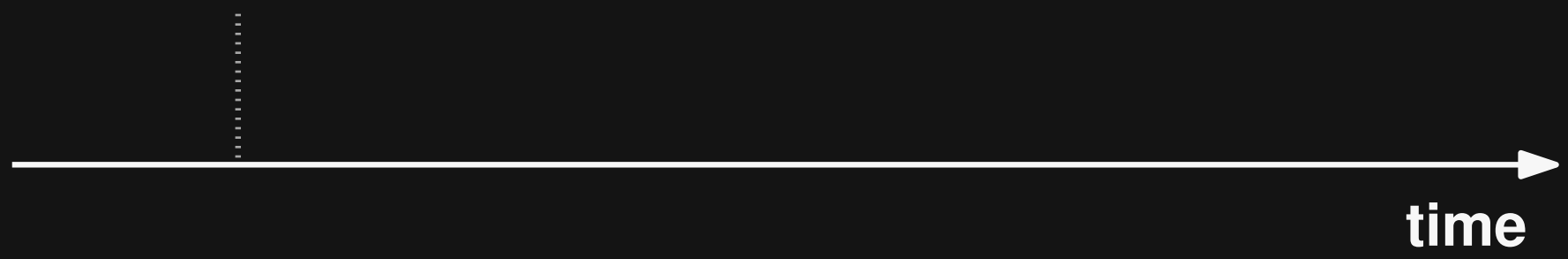
  LExecutor: Learning-Guided Execution, FSE'23

Types in Python

Typical evolution of a Python project:

Code without type annotations

```
def f(x, y):
  s = x + y
  if (s % 2) == 0:
    return True
```

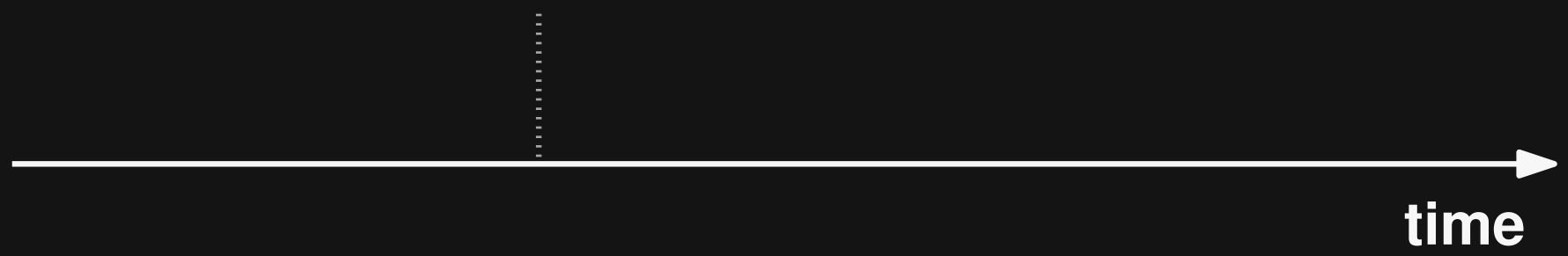


Types in Python

Typical evolution of a Python project:

Partially annotated code

```
def f(x, int, y) -> bool:
  s: int = x + y
  if (s % 2) == 0:
    return True
```



Types in Python

Typical evolution of a Python project:

Partially annotated code

```
def f(x: int, y) -> bool:
  s: int = x + y
  if (s % 2) == 0:
    return True
```

Type error!

time

Types in Python

Typical evolution of a Python project:

Fixed type error

```
def f(x: int, y) -> Optional[bool]:
  s: int = x + y
  if (s % 2) == 0:
    return True
```

time

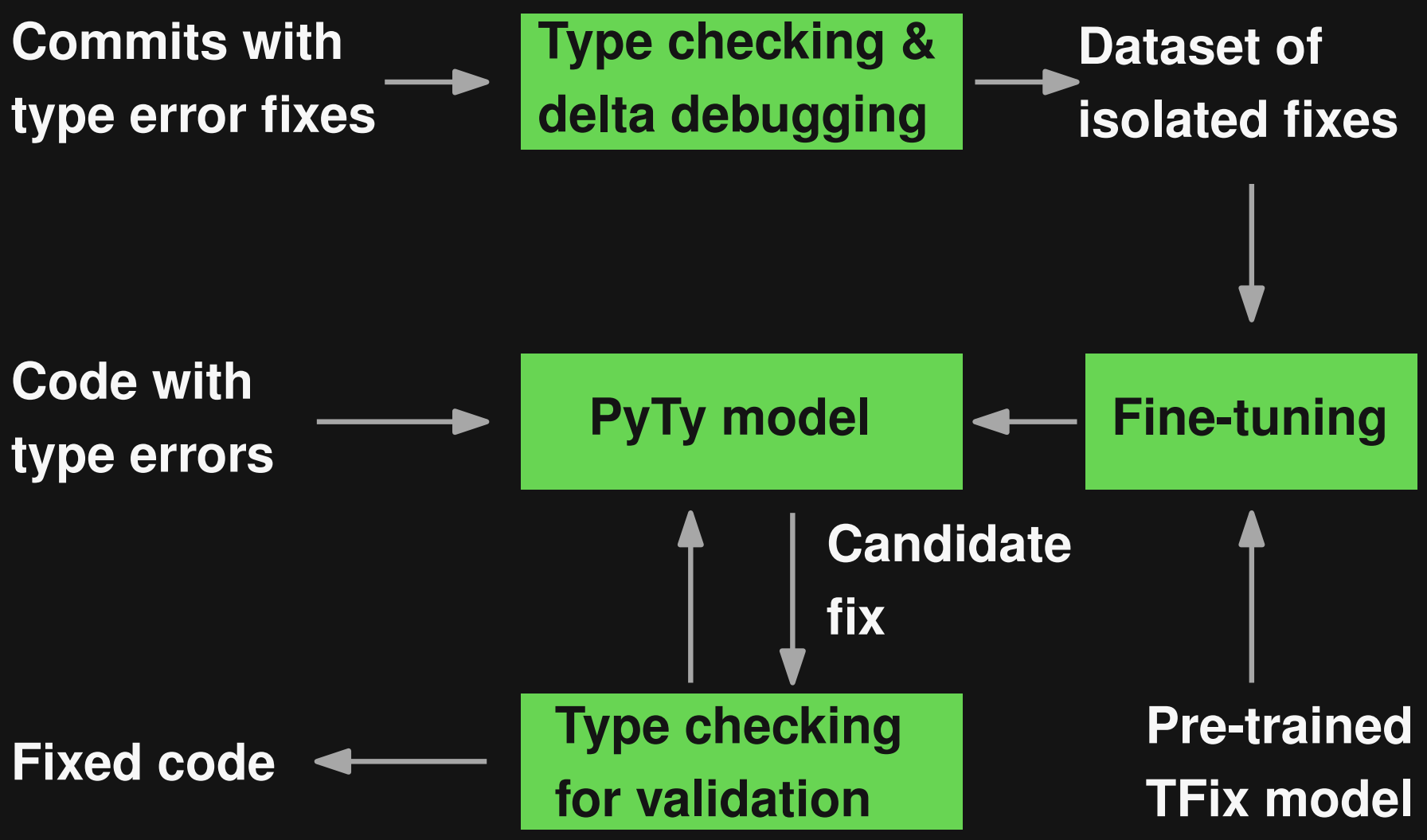
Too Many Type Errors

- Most existing Python code bases: Plenty of static type errors
- Easy to detect by gradual type checker
- But: No time to fix them all

Preliminary Study

- **Many recurring fix patterns**
  - But: No unambiguous repair rules
- **Most fixes are local, e.g., single-line**
- **Type checker helps localize fix location**

PyTy: Approach



Data Gathering

1) Keyword-based search for commits

↓

2) Type check old and new code

↓ 32k type errors removed in 4.5k commits

3) Isolate fixes of exactly one type error

↓ 2.8k isolated type error fixes

Data Gathering: Example

Error: Unbound name `basestring`

Old code:

```
# Hunk H1
class CacheKey(basestring):

# Hunk H2
  pass



# Hunk H3
  if isinstance(key, CacheKey):
    key = CacheKey(smart_str(key))

# Hunk H4
  if timeout == 0:
```

New code:

```
# Hunk H1
class CacheKey(object):

# Hunk H2
  def __init__(self, key):
    self._key = key
    ...

# Hunk H3
  if not isinstance(key, CacheKey):
    key = CacheKey(key)

# Hunk H4
  if timeout is None:
    ...
```

Data Gathering: Example

Error: Unbound name `basestring`

Old code:

```
# Hunk H1
class CacheKey(basestring):

# Hunk H2
  pass



# Hunk H3
  if isinstance(key, CacheKey):
    key = CacheKey(smart_str(key))

# Hunk H4
  if timeout == 0:
```

New code:

```
# Hunk H1
class CacheKey(object):

# Hunk H2
  def __init__(self, key):
    self._key = key
    ...

# Hunk H3
  if not isinstance(key, CacheKey):
    key = CacheKey(key)

# Hunk H4
  if timeout is None:
    ...
```

Data Gathering: Example

Error: Unbound name `basestring`

Old code:

```
# Hunk H1
class CacheKey(basestring):

# Hunk H2
  pass
```

New code:

```
# Hunk H1
class CacheKey(object):

# Hunk H2
  def __init__(self, key):
    self._key = key
    ...
```

```
# Hunk H3
  if isinstance(key, CacheKey):
    key = CacheKey(smart_str(key))

# Hunk H4
  if timeout == 0:
```

```
# Hunk H3
  if not isinstance(key, CacheKey):
    key = CacheKey(key)

# Hunk H4
  if timeout is None:
    ...
```

Data Gathering: Example

Error: Unbound name `basestring`

Old code:

```
# Hunk H1
class CacheKey(basestring):

# Hunk H2
  pass
```

New code:

```
# Hunk H1
class CacheKey(object):

# Hunk H2
  def __init__(self, key):
    self._key = key
    ...
```

```
# Hunk H3
  if isinstance(key, CacheKey):
    key = CacheKey(smart_str(key))

# Hunk H4
  if timeout == 0:
```

```
# Hunk H3
  if not isinstance(key, CacheKey):
    key = CacheKey(key)

# Hunk H4
  if timeout is None:
    ...
```

Data Gathering: Example

Error: Unbound name `basestring`

Old code:

```
# Hunk H1
class CacheKey(basestring):

# Hunk H2
  pass

# Hunk H3
  if isinstance(key, CacheKey):
    key = CacheKey(smart_str(key))

# Hunk H4
  if timeout == 0:
```

New code:

```
# Hunk H1
class CacheKey(object):

# Hunk H2
  def __init__(self, key):
    self._key = key
    ...

# Hunk H3
  if not isinstance(key, CacheKey):
    key = CacheKey(key)

# Hunk H4
  if timeout is None:
    ...
```

Data Gathering: Example

Error: Unbound name `basestring`

Old code:

```
# Hunk H1
class CacheKey(basestring):

# Hunk H2
  pass

# Hunk H3
  if isinstance(key, CacheKey):
    key = CacheKey(smart_str(key))

# Hunk H4
  if timeout == 0:
```

New code:

```
# Hunk H1
class CacheKey(object):

# Hunk H2
  def __init__(self, key):
    self._key = key
    ...

# Hunk H3
  if not isinstance(key, CacheKey):
    key = CacheKey(key)

# Hunk H4
  if timeout is None:
    ...
```

Model

Erroneous code with context → **Seq-to-seq model *** → **Fixed code**

$$fix \;\; t \;\; m \;\; l_k : \; C \qquad\qquad C'$$

Kind of type error | Error message | Line with type error | Code tokens

*** Fine-tuned model, based on pre-trained TFix** [Berabi'21]**, which is based on pre-trained T5** [Raffel'20]

PyTy: Effectiveness

Classes of type errors	Samples (test set)		Effectiveness of PyTy	
			Error removal	Exact match
Incompatible variable type	821	(83)	90.4%	65.1%
Incompatible parameter type	600	(60)	80.0%	36.7%
Incompatible return type	296	(30)	73.3%	43.3%
Invalid type	291	(30)	100.0%	83.3%
Unbound name	258	(26)	76.9%	42.3%
Incompatible attribute type	258	(26)	92.3%	73.1%
Unsupported operand	124	(13)	76.9%	38.5%
Strengthened precondition	59	(6)	83.3%	50.0%
Weakened postcondition	51	(6)	50.0%	0.0%
Call error	8	(1)	100.0%	100.0%
Total	2,766	(281)	85.4%	54.4%

Examples

Code with type error:

```
vprint(f"{prefix} {lineno}: {action_name}
  Constrain Mouse: {'yes' if constraint > 0
  else ('no' if constrained == 0 else 'check stack')}")
```

Unbound name

PyTy finds exactly the developer fix:

```
vprint(f"{prefix} {lineno}: {action_name}
  Constrain Mouse: {'yes' if constraint > 0
  else ('no' if constraint == 0 else 'check stack')}")
```

Examples

Code with type error:

Declared to have type `str`
but used as `bytes`

```
string = _fmt(string)
return lib.TCOD_console_get_height_rect_fmt(
  self.console_c, x, y, width, height, string
)
```

PyTy finds a valid fix:

```
byte_string = _fmt(string)
return lib.TCOD_console_get_height_rect_fmt(
  self.console_c, x, y, width, height, byte_string
)
```

Developer fix (semantically equivalent):

```
return lib.TCOD_console_get_height_rect_fmt(
  self.console_c, x, y, width, height, _fmt(string)
)
```

Three Examples

- **Fixing type errors with PyTy**

  PyTy: Repairing Static Type Errors in Python, FSE'23 (major rev.)

- **Neural bug detection with CMI-Finder**

  When to Say What: Learning to Find Condition-Message Inconsistencies, ICSE'23

- **Enabling execution with LExecutor**

  LExecutor: Learning-Guided Execution, FSE'23 (major rev.)

Motivation

Example 1:

```
if len(bits) != 4 or len(bits) != 6:
  raise template.TemplateSyntaxError("%r takes
    exactly four or six arguments (second argument
    must be 'as')" % str(bits[0]))
```

Motivation

Example 1:

Always `True`

Doesn't match the message

```
if len(bits) != 4 or len(bits) != 6:
  raise template.TemplateSyntaxError("%r takes
    exactly four or six arguments (second argument
    must be 'as')" % str(bits[0]))
```

Motivation

Example 1:

Always `True`

Doesn't match the message

```
if len(bits) != 4 or len(bits) != 6:
  raise template.TemplateSyntaxError("%r takes
    exactly four or six arguments (second argument
    must be 'as')" % str(bits[0]))
```

Example 2:

```
if n2 > n1 :
  raise ValueError('Total internal reflection
    impossible for n1 > n2')
```

Motivation

Example 1:

Always `True`

Doesn't match the message

```
if len(bits) != 4 or len(bits) != 6:
  raise template.TemplateSyntaxError("%r takes
    exactly four or six arguments (second argument
    must be 'as')" % str(bits[0]))
```

Example 2:

Condition and message are inconsistent

```
if n2 > n1 :
  raise ValueError('Total internal reflection
    impossible for n1 > n2')
```

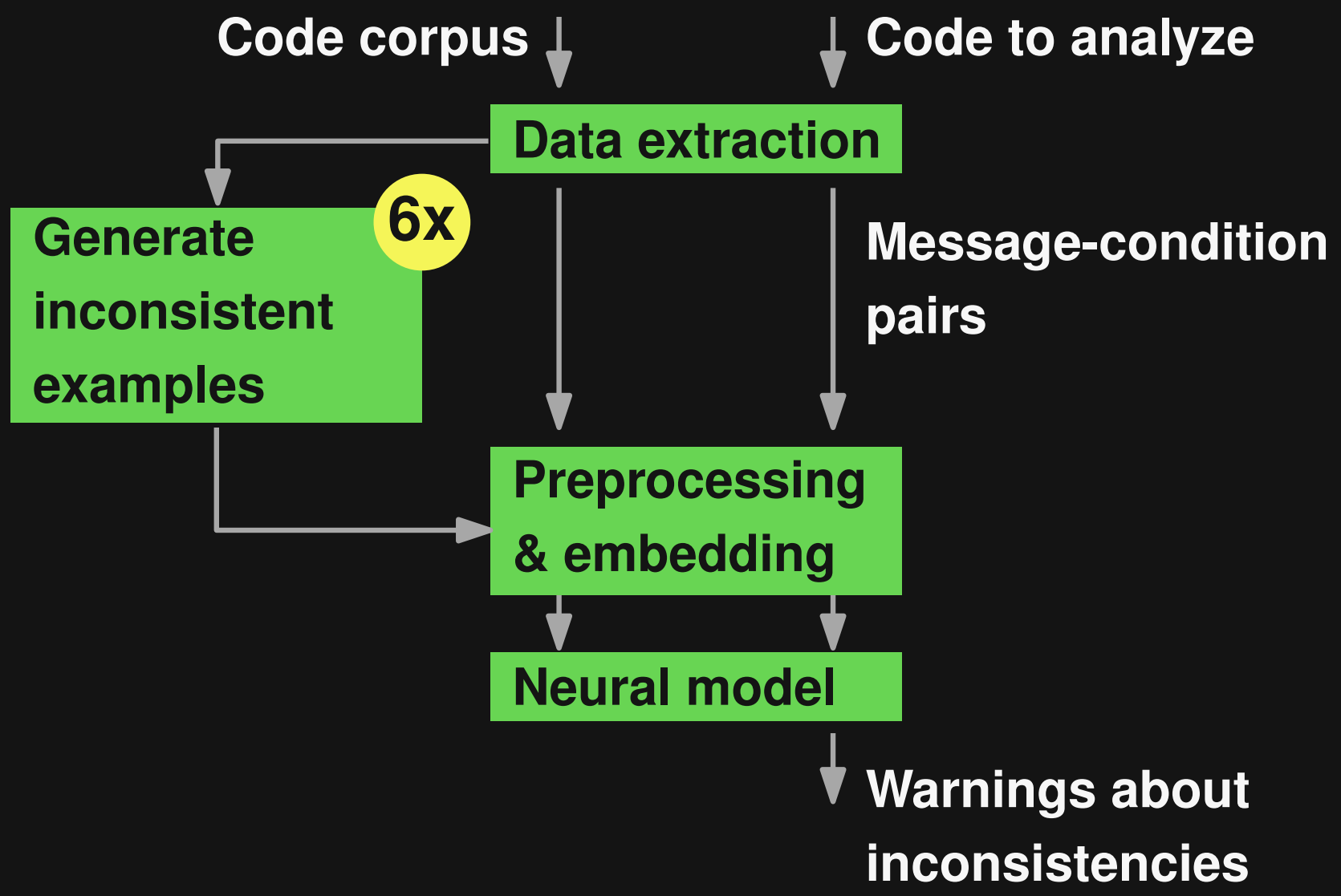
CMI-Finder

Goal:

Detect condition-message inconsistencies

- Why?
  - Incorrect conditions may raise unnecessary warnings or suppress expected warnings
  - Incorrect messages make debugging unnecessarily hard
- Hard problem!
  - Must understand both NL and PL

Overview of CMI-Finder



Training

Prediction

Generating Inconsistent Examples

Six generation strategies

- Mutation of operators
- Mutation of error messages
- Random re-combination
- Pattern-based mutation
- Embedding-based token replacement
- Language model-based generation of error message

Generating Inconsistent Examples

Six generation strategies

- Mutation of operators
- Mutation of error messages
- Random re-combination
- Pattern-based mutation
- Embedding-based token
- Language model-based g
  message

Example:

```
if result.status in (0, 3):
  log.warning("Invalid status")
```

⇓

```
if result.status in (0, 3):
  log.warning("Valid status")
```

Generating Inconsistent Examples

Six generation strategies

- Mutation of operators
- Mutation of error mess
- Random re-combinatio
- Pattern-based mutatior
- Embedding-based token replacement
- Language model-based generation of error message

Example:

```
if not isinstance(config, (tuple, list)):
  raise TypeError('Unable to decode
    config: {}'.format(config))
```

⇓

```
if not isinstance(config, (tuple, list)):
  raise ValueError('Unable to decode
    config: {}'.format(config))
```

Generating Inconsistent Examples

Six generation strategies

- Mutation of operators
- Mutation of error mes
- Random re-combinati
- Pattern-based mutatic
- Embedding-based token replacement
- Language model-based generation of error message

Example:

```
if x == 0:
  raise ValueError('x must not be zero')
```

⇓

```
if x != 0:
  raise ValueError('x cannot be lower than 0')
```

Train & Predict

Fine-tuned CodeT5 model

tokenize(condition) ⊕ tokenize(message) → **Model** → "consistent" or "inconsistent"

Also tried, but less effective:

- Binary classifier
- Contrastive learning

Evaluation

- **Training data**
  - 300k pairs from 40k Python projects
    + 300k inconsistent pairs
- **Real-world test data**
  - 66 pairs from 33 historic fixes of condition-message inconsistencies
  - Seven previously unseen Python projects

Results

- **AUC of 0.91 (synthetic data) and 0.82 (real-world data)**
  - E.g., 0.78 precision and 0.72 recall on historic fixes
- **50 new inconsistencies in held-out projects**
- **Complements flake8 and outperforms a GPT-3 baseline**

Three Examples

- **Fixing type errors with PyTy**

  PyTy: Repairing Static Type Errors in Python, FSE'23 (major rev.)

- **Neural bug detection with CMI-Finder**

  When to Say What: Learning to Find Condition-Message Inconsistencies, ICSE'23

- **Enabling execution with LExecutor** ←

  LExecutor: Learning-Guided Execution, FSE'23 (major rev.)

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 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]

# ...
```

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]

# ...
```

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?

Enables various dynamic analyses

- Check for exceptions and assertion violations
- Compare two code snippets for semantic equivalence
- Validate static analysis warnings
- Validate and filter LLM-predicted code
- $\langle Your\ favorite\ application\ here \rangle$

Executing Ain't Easy

Lots of incomplete code:

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

Executing Ain't Easy

Lots of incomplete code:

- Code snippets from Stack Overflow
- Code generated by language models
- Code extracted from deep inside 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

Underconstrained execution:
No guarantee that values are realistic

LExecutor: Learning-Guided Execution, FSE'23

Example

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

Let's "lexecute" the motivating example:

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]

# ...
```

Example

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]

# ...
```

Example

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

Let's "lexecute" the motivating example:

Function that returns `True`

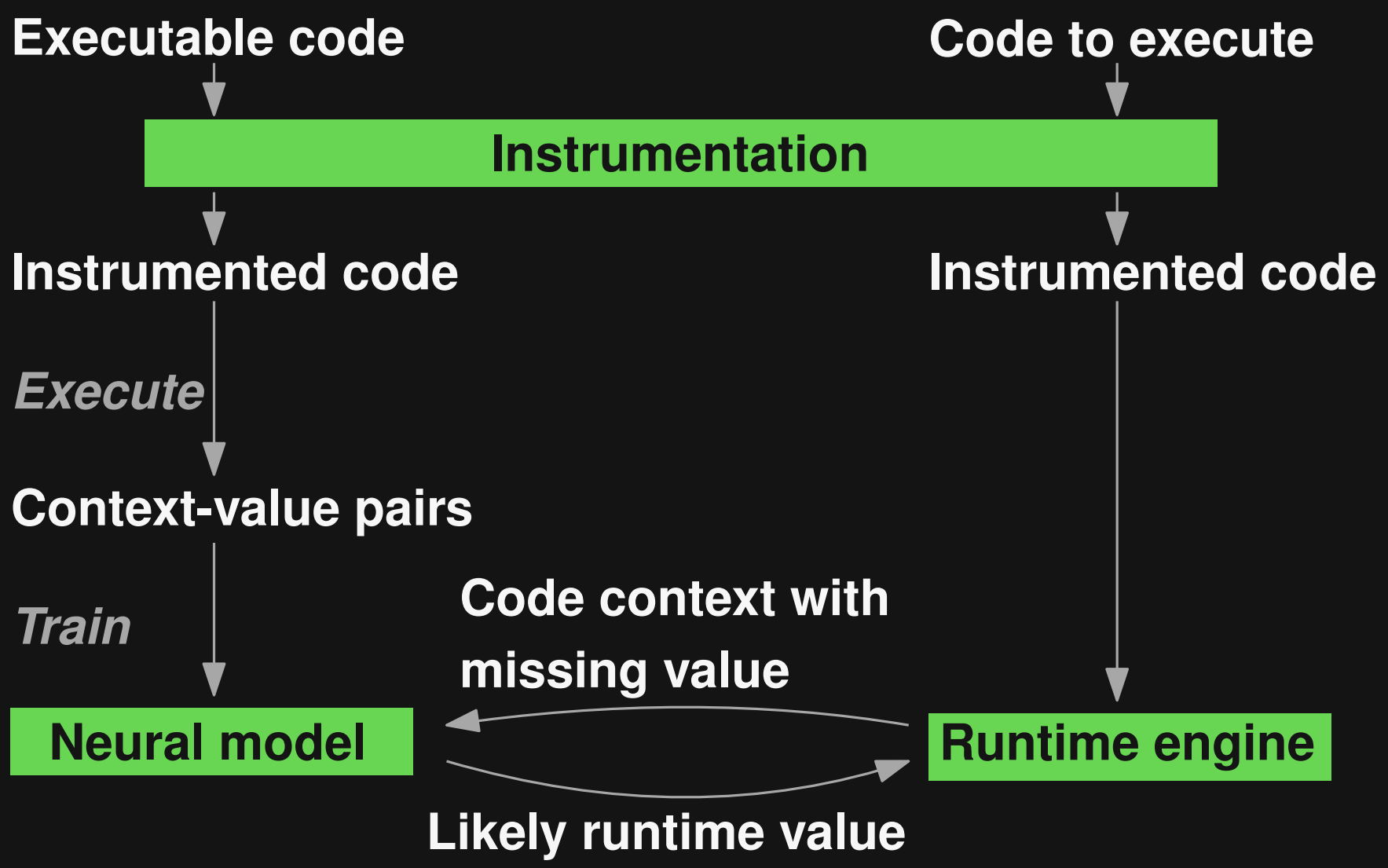
Non-empty list

```
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



Neural Model: Data Representation

Neural Model: Data Representation

Code context → **Model** → **Value**

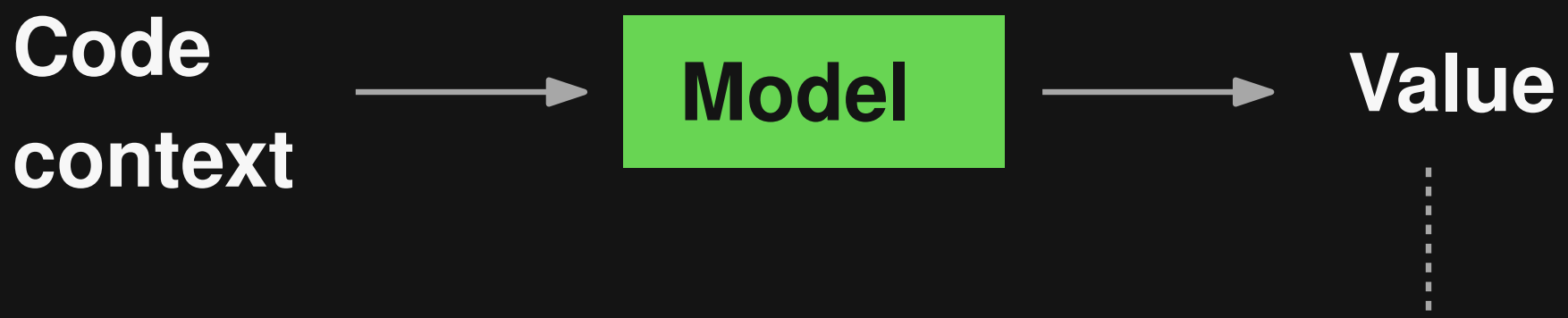
$$n \ \langle sep \rangle \ k \ \langle sep \rangle \ c_{pre} \ \langle mask \rangle \ c_{post}$$

Name used to refer to a value

Kind of value (variable, attribute, or return value)

Code before/after the reference to the value

Neural Model: Data Representation



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

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

Train & Predict

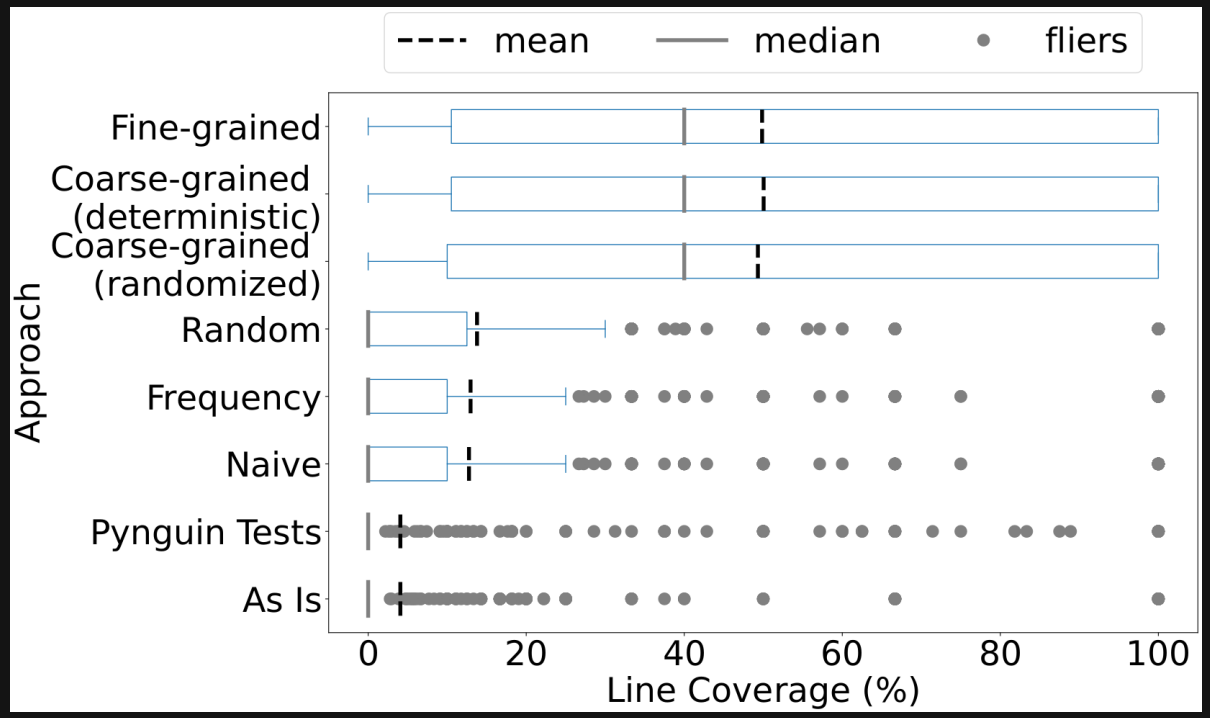
- **Fine-tune a pre-trained CodeT5 model**
- **During prediction:**
  For each use of a value
  - Read value and, if it exists, return it
  - If undefined, query the model and return its prediction

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

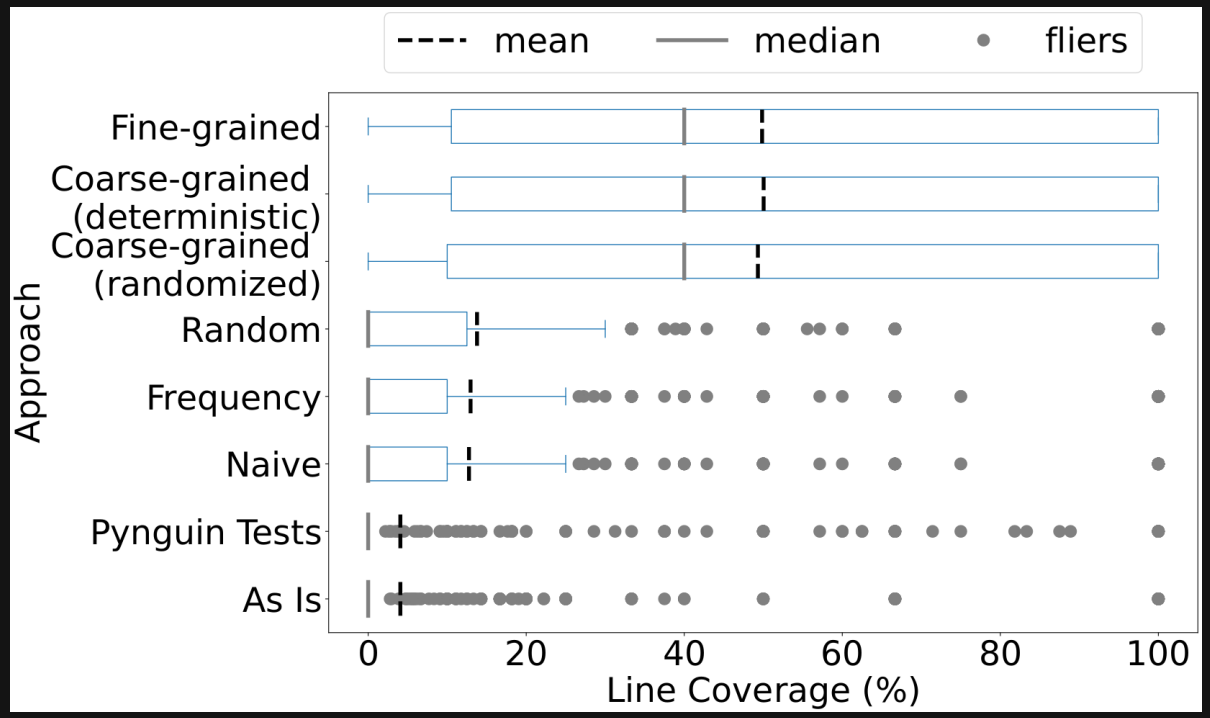
Results

- **Accuracy** of neural model:
  80.1% (top-1) – 94.2% (top-5)
- **Successfully executed lines**:



Results

- **Accuracy** of neural model:
  80.1% (top-1) – 94.2% (top-5)
- **Successfully executed lines**:



Variants of LExecutor

LExecutor without model

State of the art

Example: Stack Overflow Snippet

```
plt.figure(figsize=(16, 8))
for i in range(1, 7):
  plt.subplot(2, 3, i)
  plt.title('Histogram of {}'.format(str(i)))
  plt.hist(x[:, i-1], bins=60)
```

Example: Stack Overflow Snippet

Object

Method that returns nothing

```
plt.figure(figsize=(16, 8))
for i in range(1, 7):
  plt.subplot(2, 3, i)
  plt.title('Histogram of {}'.format(str(i)))
  plt.hist(x[:, i-1], bins=60)
```

Methods that return nothing

Non-empty tuple

Example: Stack Overflow Snippet

Object

Method that returns nothing

```
plt.figure(figsize=(16, 8))
for i in range(1, 7):
  plt.subplot(2, 3, i)
  plt.title('Histogram of {}'.format(str(i)))
  plt.hist(x[:, i-1], bins=60)
```

Crash

TupleError: tuple indices must be integers or slices, not tuple

Methods that return nothing

Non-empty tuple

Summary

- **Machine learning for programming**
  - Fixing type errors with PyTy
  - Neural bug detection with CMI-Finder
  - Enabling execution with LExecutor
- **Next steps**
  - Check topics on course page
  - Indicate your preferences by Oct 26