Neural Software Analysis:
Recent Advances on Types, Bugs, and Executions

Michael Pradel (University of Stuttgart)
Joint work with Beatriz Souza, Islem Bouzenia, Luca Di Grazia, and Wai Chow
Program Analysis
How to Create a Program Analysis?
Traditional program analysis

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

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, 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

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

Neural software analysis

- Automatically learned within hours or days
- Data-driven prediction
- Learn instead of hard-code heuristics
- Use big code to our benefit
This Talk

- **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.)*
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
Typical evolution of a Python project:

Partially annotated code

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

The Evolution of Type Annotations in Python: An Empirical Study, FSE'22
Typical evolution of a Python project:

Partially annotated code

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

Type error!
Types in Python

Typical evolution of a Python project:

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

Fixed type error
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

The Evolution of Type Annotations in Python: An Empirical Study, FSE’22
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: Repairing Static Type Errors in Python, FSE’23 (major rev.)
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
Error: Unbound name `basestring`

Old code:

```python
# 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:

```python
# 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:
```

`PyTy: Repairing Static Type Errors in Python, FSE’23 (major rev.)`
Error: Unbound name `basestring`

Old code:

```python
# 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:

```python
# 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:
    ...
```

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

## Error: Unbound name `basestring`

**Old code:**

```python
# 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:**

```python
# 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:
```

PyTy: Repairing Static Type Errors in Python, FSE’23 (major rev.)
Data Gathering: Example

Error: Unbound name `basestring`

Old code:

```python
# 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:

```python
# 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:
```
Error: Unbound name `basestring`

<table>
<thead>
<tr>
<th>Old code:</th>
<th>New code:</th>
</tr>
</thead>
</table>
| # Hunk H1  
class CacheKey(basestring):
# Hunk H2
    pass
# Hunk H3
    if isinstance(key, CacheKey):
        key = CacheKey(smart_str(key))
# Hunk H4
    if timeout == 0:
| # 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:
        ...

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

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

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

\[ \text{fix } t \cdot m \cdot l_k : C \]

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

\[ C' \]
<table>
<thead>
<tr>
<th>Classes of type errors</th>
<th>Samples (test set)</th>
<th>Effectiveness of PyTy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incompatible variable type</td>
<td>821 (83)</td>
<td>90.4% 65.1%</td>
</tr>
<tr>
<td>Incompatible parameter type</td>
<td>600 (60)</td>
<td>80.0% 36.7%</td>
</tr>
<tr>
<td>Incompatible return type</td>
<td>296 (30)</td>
<td>73.3% 43.3%</td>
</tr>
<tr>
<td>Invalid type</td>
<td>291 (30)</td>
<td>100.0% 83.3%</td>
</tr>
<tr>
<td>Unbound name</td>
<td>258 (26)</td>
<td>76.9% 42.3%</td>
</tr>
<tr>
<td>Incompatible attribute type</td>
<td>258 (26)</td>
<td>92.3% 73.1%</td>
</tr>
<tr>
<td>Unsupported operand</td>
<td>124 (13)</td>
<td>76.9% 38.5%</td>
</tr>
<tr>
<td>Strengthened precondition</td>
<td>59 (6)</td>
<td>83.3% 50.0%</td>
</tr>
<tr>
<td>Weakened postcondition</td>
<td>51 (6)</td>
<td>50.0% 0.0%</td>
</tr>
<tr>
<td>Call error</td>
<td>8 (1)</td>
<td>100.0% 100.0%</td>
</tr>
<tr>
<td>Total</td>
<td>2,766 (281)</td>
<td>85.4% 54.4%</td>
</tr>
</tbody>
</table>
Examples

Code with type error:

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

PyTy finds exactly the developer fix:

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

Unbound name
Examples

Code with **type error**:

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

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

**PyTy finds a valid fix:**

```python
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):**

```python
return lib.TCOD_console_get_height_rect_fmt(
    self.console_c, x, y, width, height, __fmt(string)
)
```
This Talk

- **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:

```python
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

if \text{len(bits)} \neq 4 \text{ or } \text{len(bits)} \neq 6:
raise \text{template.TemplateSyntaxError}("%r takes exactly four or six arguments (second argument must be 'as')" % \text{str(bits[0])})

Doesn't match the message

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

Example 1:  

```python
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]))
```

Always True

Example 2:  

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

Example 2:

Doesn't match the message
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]))

Always True

Example 2:

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

Condition and message are inconsistent

Doesn’t match the message
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

- Code corpus
- Code to analyze
- Data extraction
- Message-condition pairs
- Preprocessing & embedding
- Neural model
- Warnings about inconsistencies

Generate inconsistent examples

6x

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

Example:

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

Example:

```python
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
This Talk

- **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.)*
Imagine you want to execute this code:

```python
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]

# ...
```
Imagine you want to execute this code:

```python
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**

**Missing variable**
Motivation

Imagine you want to execute this code:

```python
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 function**
```
has_min_size
```

**Missing variable**
```
len
```
Motivation

Imagine you want to execute this code:

```python
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 function

```python
has_min_size(all_data)
```

Missing variable

```python
train_len
```

Missing variable

```python
config_str
```
Motivation

Imagine you want to execute this code:

```python
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]
```

# ...
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
- ⟨Your favorite application here⟩

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

Let’s “lexecute” the motivating example:

```python
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]

# ...
```
Let’s “lexecute” the motivating example:

```python
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]

# ...
```
Let’s “lexecute” the motivating example:

Function that returns True

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

Non-empty list

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

# ...
Let’s “lexecute” the motivating example:

```python
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 list
Non-empty string
Function that returns True

Non-empty string
Example

Let’s “lexecute” the motivating example:

Function that returns True

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

Non-empty list

```python
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

```python
# ...
```

LExecutor: Learning-Guided Execution, FSE’23 (major rev.)
Overview of LExecutor

Executable code

Instrumented code

Instrumentation

Code to execute

Instrumented code

Execute

Context-value pairs

Train

Neural model

Code context with missing value

Runtime engine

Likely runtime value

Training

Prediction
Neural Model: Data Representation

Code context → Model → Value

LExecutor: Learning-Guided Execution, FSE’23 (major rev.)
Neural Model: Data Representation

\[ n \langle \text{sep} \rangle k \langle \text{sep} \rangle c_{\text{pre}} \langle \text{mask} \rangle c_{\text{post}} \]

- **Name used to refer to a value**
- **Kind of value** (variable, attribute, or return value)
- **Code** before/after the reference to the value
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:**

![Diagram showing line coverage for different approaches.]
Results

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

- **Successfully executed lines:**

![Graph showing line coverage for different approaches]

**Variants of LExecutor**
- LExecutor without model
- State of the art
Example: Stack Overflow Snippet

```python
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

```python
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)
```

Object

```
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)
```

Method that returns nothing

```python
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)
```

Non-empty tuple

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

LExecutor: Learning-Guided Execution, FSE’23 (major rev.)
The Road Ahead

Papers on neural software analysis *

* Estimate based on Neural Software Analysis, CACM’22
The Road Ahead
The Road Ahead

General-purpose language models
The Road Ahead

General-purpose language models

Combining neural & traditional analysis

2015  2020
The Road Ahead

General-purpose language models

Combining neural & traditional analysis

Reasoning about executions
Summary

- **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.)*

Thanks!