When to Say What:

Learning to Find Condition-Message Inconsistencies

Islem Bouzenia, Michael Pradel

Software Lab – University of Stuttgart

Motivating Example

What's wrong with this code?

```
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]))
```

Motivating Example

What's wrong with this code?

```
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]))
```

Condition and message are inconsistent!

(The condition is always true.)

Problem

Finding condition-message inconsistencies

Problem

Any statement that emits a message
(e.g., raising exception, printing, logging)

Finding condition-message inconsistencies

Problem

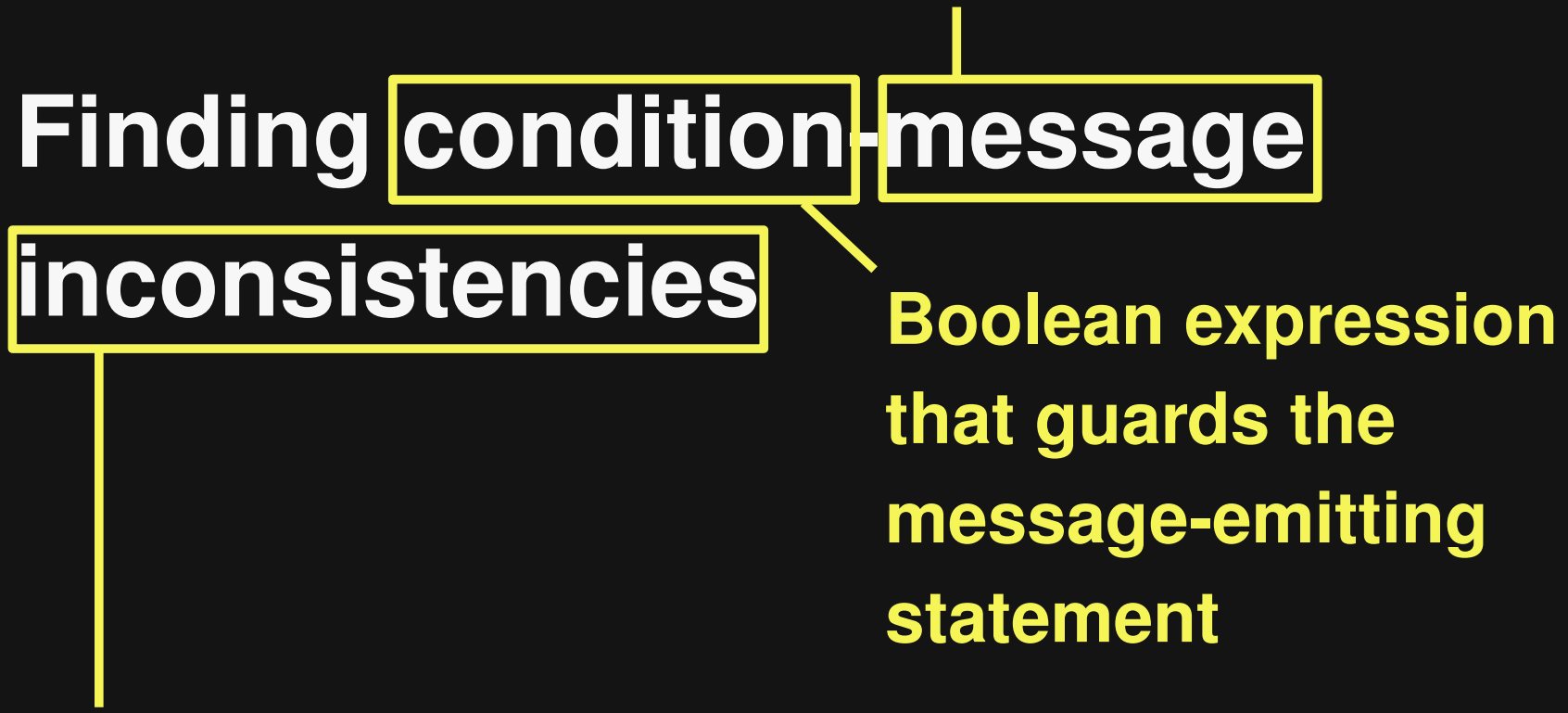
Any statement that emits a message
(e.g., raising exception, printing, logging)

Finding condition-message inconsistencies

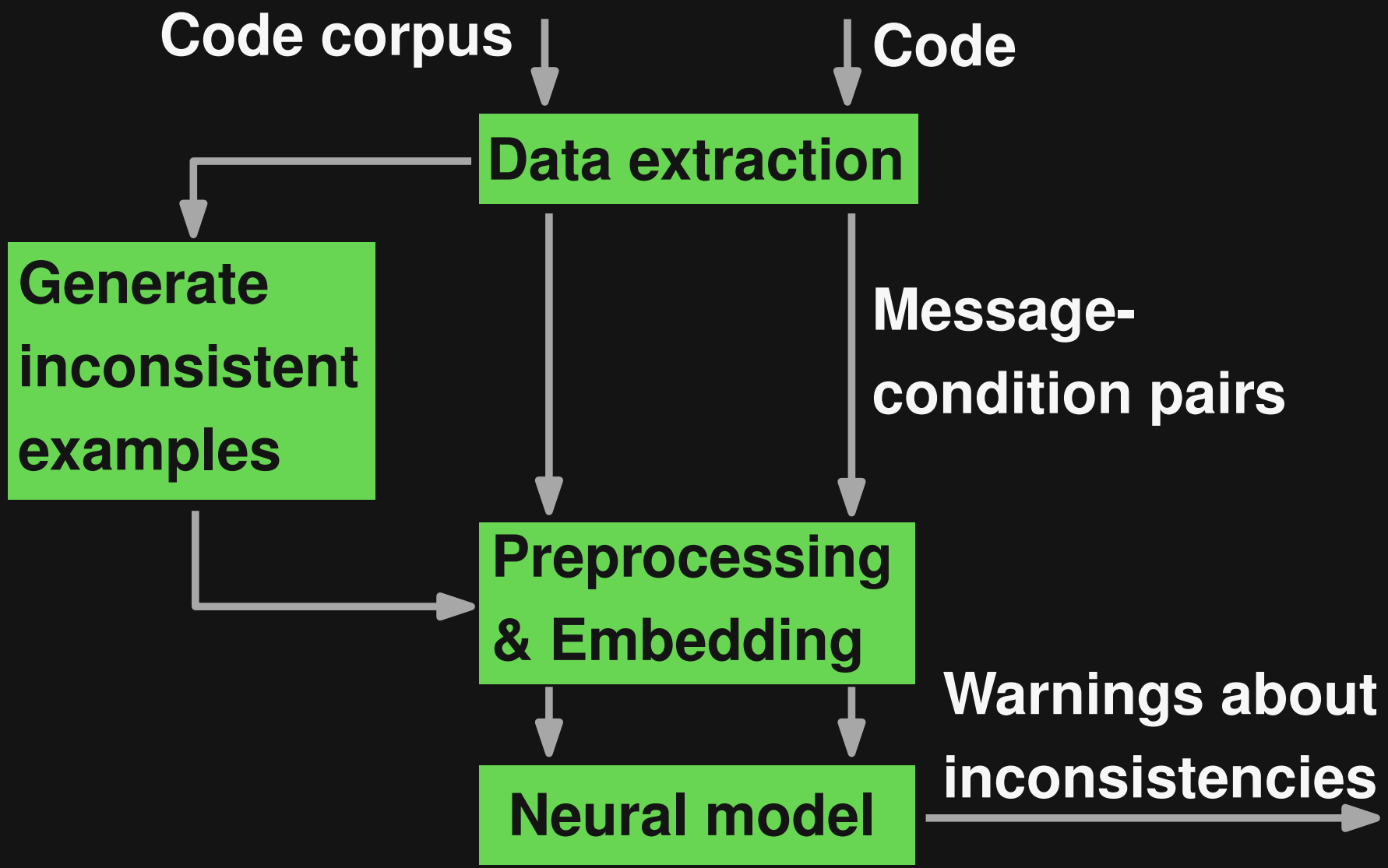
Boolean expression that guards the message-emitting statement

Problem

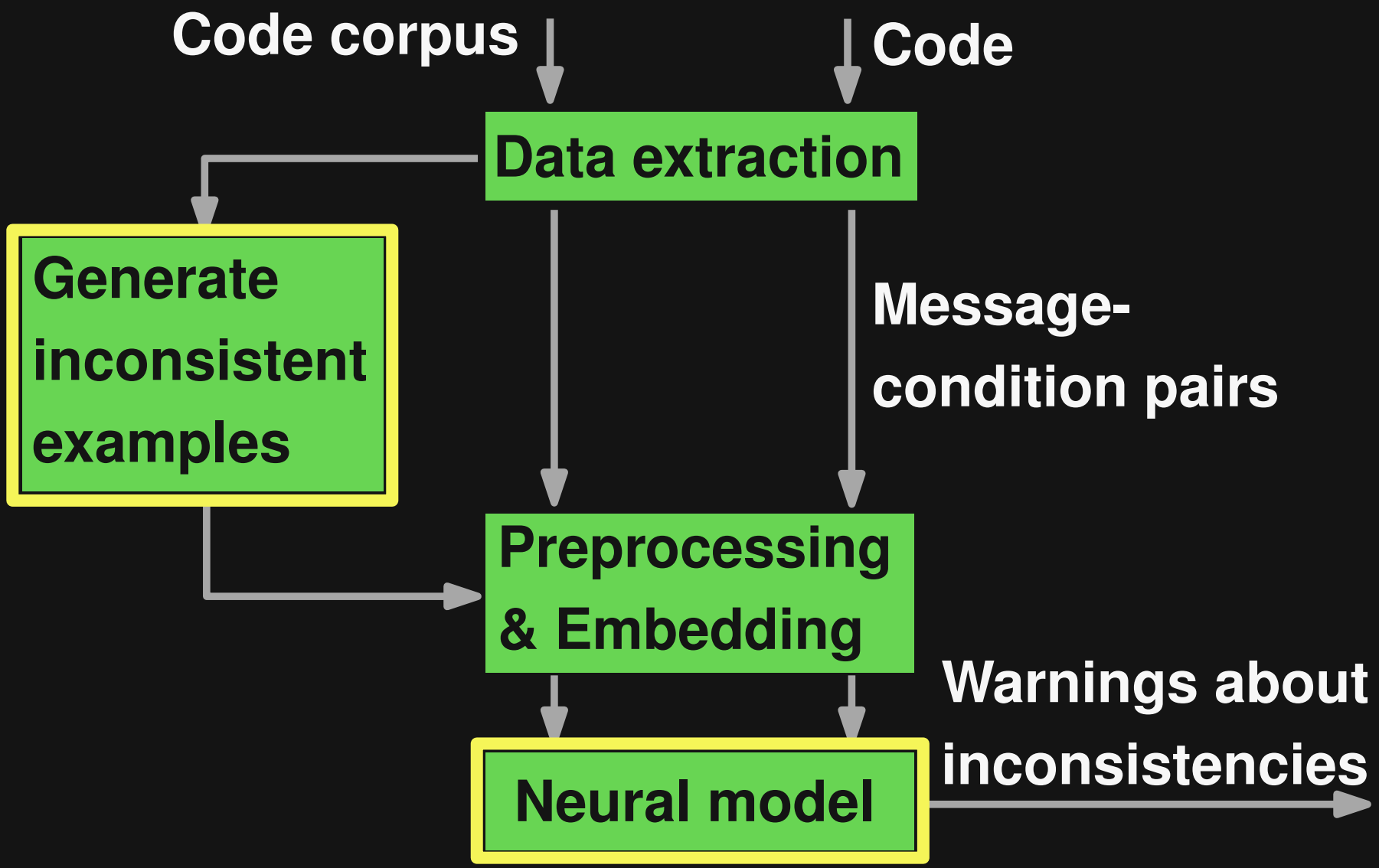
Any statement that emits a message
(e.g., raising exception, printing, logging)



CMI-Finder: Overview



CMI-Finder: Overview



Generating Inconsistent Examples

Goals: Realism, diversity, scalability

Six techniques

- Mutation of operators
- Mutation of messages
- Random re-combination
- Pattern-based mutation
- Embedding-based token replacement
- LLM-based generation of messages

Generating Inconsistent Examples

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

- Embedding-based token replacement
- LLM-based generation of messages

Generating Inconsistent Examples

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

Token with a similar embedding

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

- Embedding-based token replacement
- LLM-based generation of messages

Generating Inconsistent Examples

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

- Embedding-based token replacement
- LLM-based generation of messages

Generating Inconsistent Examples

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

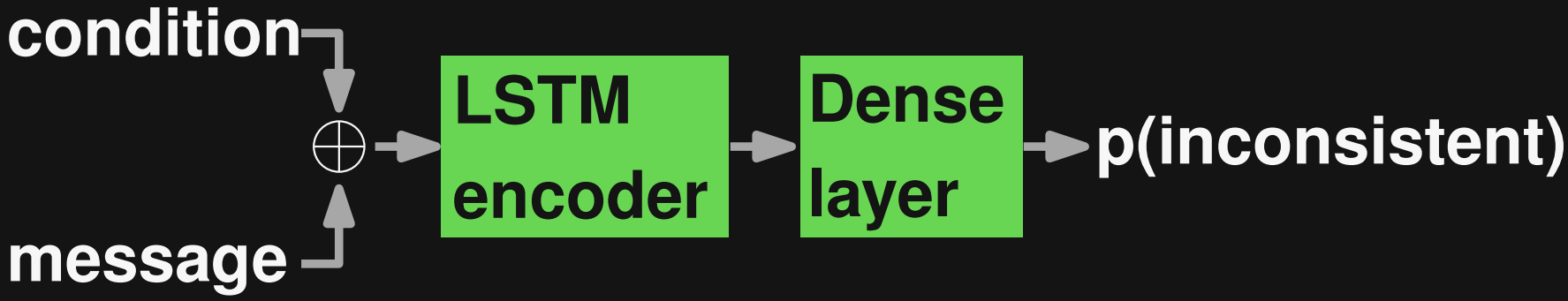
Generated by LLM for a different condition

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

- Embedding-based token replacement
- LLM-based generation of messages

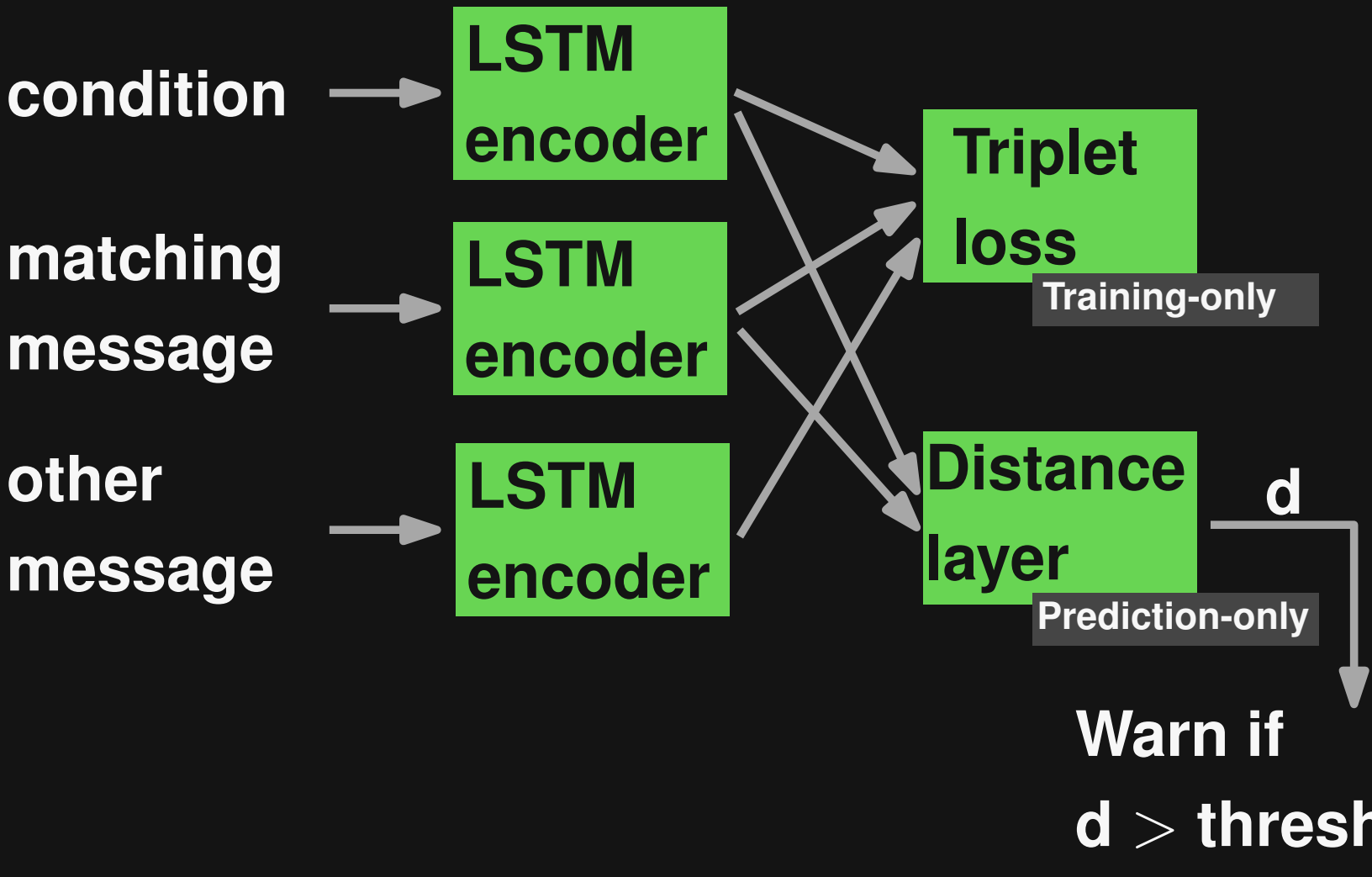
Neural Models

Option 1: Binary classification



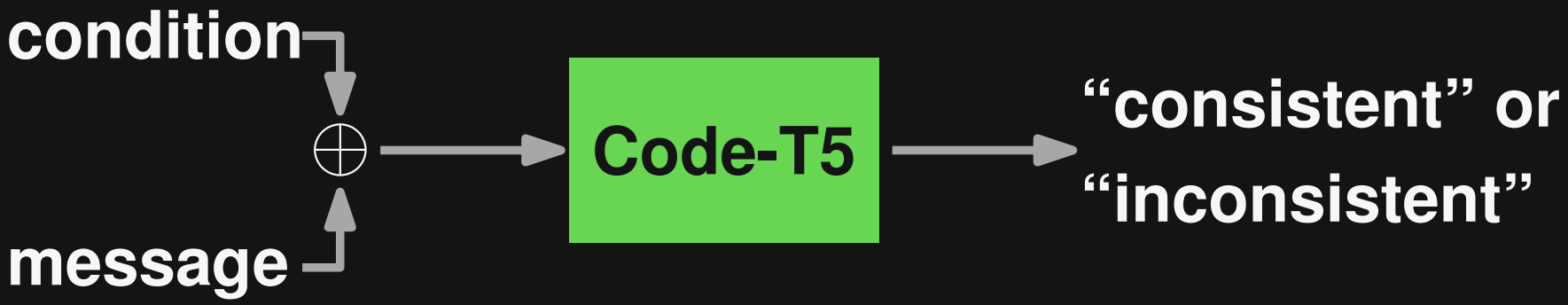
Neural Models

Option 2: Triplet loss



Neural Models

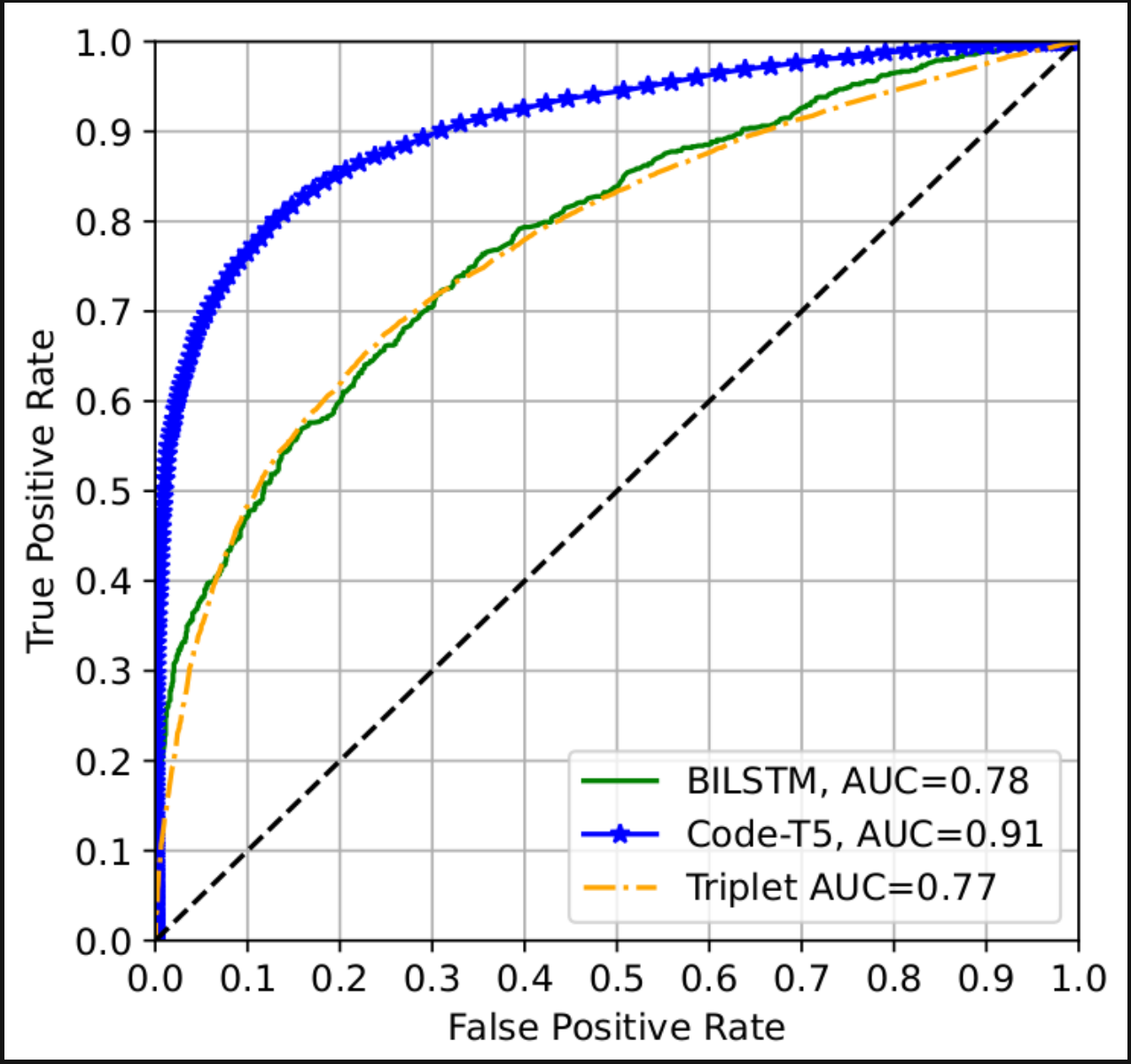
Option 3: Text-to-text transformer



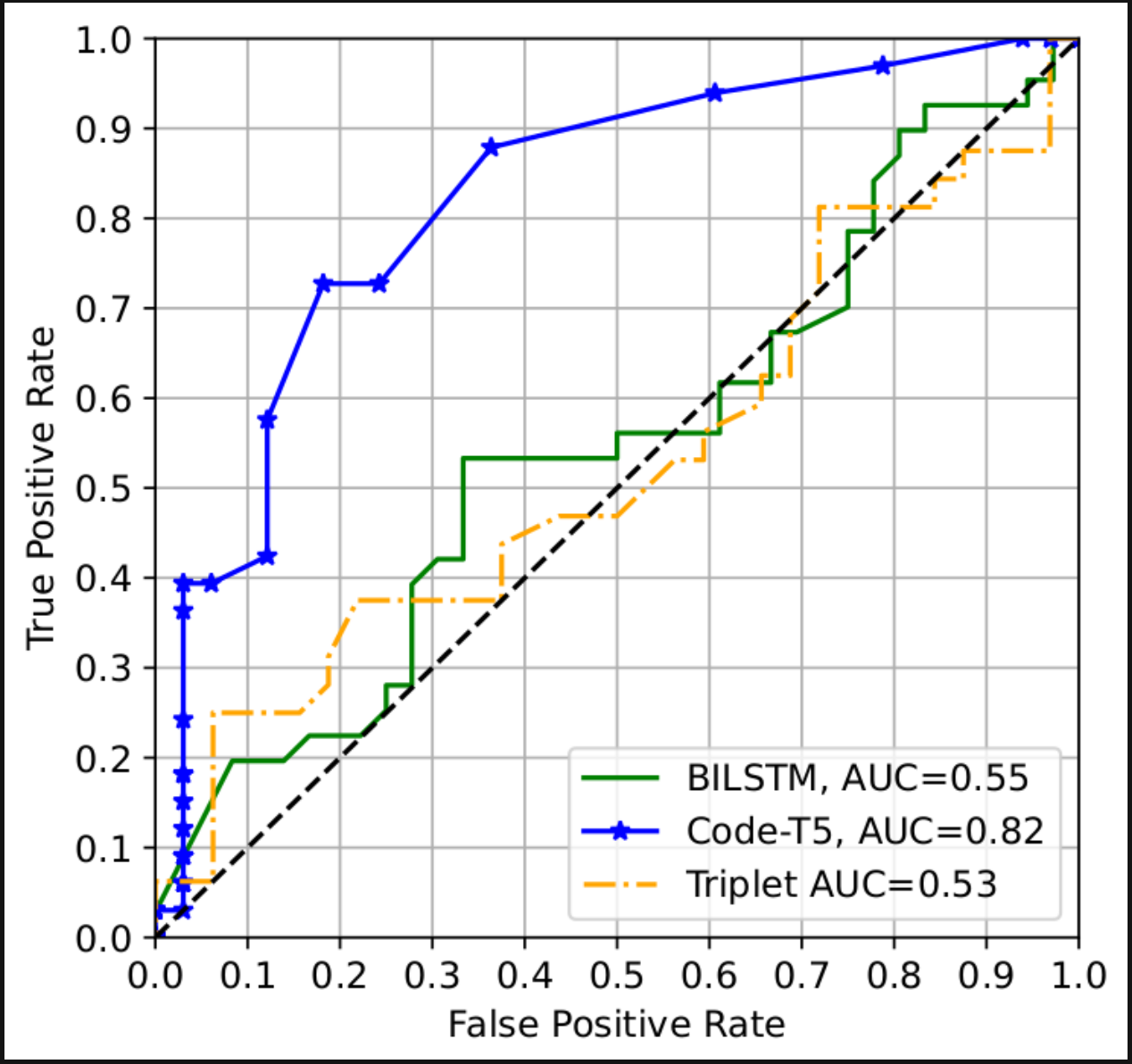
Evaluation

- **Training data:**
  600k condition-message pairs
  - 50% from 40k Python projects
  - 50% generated inconsistent examples
- **Test data**
  - 10k held-out pairs (synthetic)
  - Past, real-world bugs: 33 buggy + 33 fixed pairs
  - Seven previously unseen Python projects

Effectiveness: Synthetic Data



Effectiveness: Past, Real Bugs



Effectiveness: New Bugs

21 previously unknown inconsistencies

Examples:

```
if not isinstance(p2, PolyElement):
  raise ValueError('p1 and p2 must have
                    the same ring')
```

```
if not (os.path.isdir(tf_source_path) and
        os.path.isfile(syslibs_configure_path) and
        os.path.isfile(workspace0_path)):
  raise ValueError('The path to the TensorFlow source
                    must be passed as the first argument')
```

Both confirmed and fixed bugs, from *Simpy* and *TensorFlow*

Effectiveness: New Bugs

21 previously unknown inconsistencies

Examples:

Copy & paste mistake?

```
if not isinstance(p2, PolyElement):
  raise ValueError('p1 and p2 must have
                    the same ring')

if not (os.path.isdir(tf_source_path) and
        os.path.isfile(syslibs_configure_path) and
        os.path.isfile(workspace0_path)):
  raise ValueError('The path to the TensorFlow source
                    must be passed as the first argument')
```

Both confirmed and fixed bugs, from *Simpy* and *TensorFlow*

Effectiveness: New Bugs

21 previously unknown inconsistencies

Examples:

Copy & paste mistake?

```
if not isinstance(p2, PolyElement):
  raise ValueError('p1 and p2 must have
                    the same ring')
```

Message is incomplete

```
if not (os.path.isdir(tf_source_path) and
        os.path.isfile(syslibs_configure_path) and
        os.path.isfile(workspace0_path)):
  raise ValueError('The path to the TensorFlow source
                    must be passed as the first argument')
```

Both confirmed and fixed bugs, from *Simpy* and *TensorFlow*

More Results: Paper

- Comparison with flake8 and GPT3 bug fixing demo
- Efficiency: 10s to 1000s checks per second
- Impact of hyperparameters

When to Say What: Learning to Find Condition-Message Inconsistencies

Islem Bouzenia
University of Stuttgart
Germany
fi_bouzenia@esi.dz

Michael Pradel
University of Stuttgart
Germany
michael@binaervarianz.de

***Abstract*—Programs often emit natural language messages, e.g., in logging statements or exceptions raised on unexpected paths. To be meaningful to users and developers, the message, i.e., *what* to say, must be consistent with the condition under which it gets triggered, i.e., *when* to say it. However, checking for inconsistencies between conditions and messages is challenging because the conditions are expressed in the logic of the programming language, while messages are informally expressed in natural language. This paper presents CMI-Finder, an approach for detecting *condition-message inconsistencies*. CMI-Finder is based on a neural model that takes a condition and a message as its input and then predicts whether the two are consistent. To address the problem of obtaining realistic, diverse, and large-scale training data, we present six techniques to generate large numbers of inconsistent examples to learn from automatically. Moreover, we describe and compare three neural models, which are based on binary classification, triplet loss, and fine-tuning, respectively. Our evaluation applies the approach to 300K condition-message statements extracted from 42 million lines of Python code. The best model achieves a precision of 78% at a recall of 72% on a dataset of past bug fixes. Applying the approach to the newest versions of popular open-source projects reveals 50 previously unknown bugs, 19 of which have been confirmed by the developers so far.**

I. Introduction

Programs often emit natural language messages to inform the user about a specific event or an error occurring during the execution. These messages range from being purely informational, e.g., when logging the state of the program, to explaining why the entire execution gets terminated, e.g., when raising an exception. Code that triggers a message is typically guarded by some condition. To be meaningful to users and developers, the condition must match the message emitted by a program. In other words, *what* the program is saying should be consistent with *when* the program is saying it.

Unfortunately, not all condition-message pairs are consistent, which may harm the robustness of a program and make debugging unnecessarily difficult. There are two main reasons for condition-message inconsistencies. First, the condition may not accurately reflect when the developer intends to print a message or raise an exception. An incorrect condition may cause some noteworthy state to remain unnoticed, or perhaps even worse, cause an exception to be raised even though no unexpected state was ever reached. For example, consider the real-world example in Figure 1a. The condition is equivalent

```
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]))
```

(a) Inconsistency in the Pinax project due to an incorrect condition.

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

(b) Inconsistency in the Sympy project due to an incorrect message.

Fig. 1: Real-world examples of condition-message inconsistencies.

to `not (len(bits) == 4 and len(bits) == 6)`, which will always evaluate to `True`, while the message states that the reason for the exception is that `len(bits)` is not among the values `(4,6)`.

Second, the message may be incomplete, misleading, or even outright wrong. In this case, a user or developer may not understand the reason for a message or an exception, and as a result, perhaps even modify the code or the input in a way that introduces more bugs. Figure 1b shows a real-world example of this scenario. The exception message incorrectly claims the problem to be that `n1 > n2`, while the condition actually is raised when `n1 < n2`.

To better understand the importance of conditional messages, we perform a preliminary study of seven popular open-source projects written in Python. Analyzing the if-statements in their code shows that 20% of them output a message. In other words, there is a large number of conditions and exceptions that developers intend to be consistent, motivating a technique for automatically checking this property. We further analyze commits in these project to search for fixes of condition-message inconsistencies. The search results show that the inconsistency problem affects even popular open-source projects, such as scikit-learn (3 instances over 1,000 bug-fixing commits), Scrapy (2 instances over 1,000 bug-fixing commits), and Sympy (4 instances over 1,000 bug-fixing commits). In addition, 10% to 11% of the bug-fixing commits where a change occurs in a condition-message statement are

Conclusions

- New problem at intersection of PL/NL: Detecting condition-message inconsistencies
- Six techniques for generating likely bugs
- Neural model effective at finding bugs

https://github.com/sola-st/CMI-Finder