ChangeGuard: Validating Code Changes via Pairwise Learning-Guided Execution

Michael Pradel

University of Stuttgart

Joint work with Lars Gröninger and Beatriz Souza

SOLA SoftwareLab — University of Stuttgart Germany

erc European Research Council

Motivation

Code change meant to be
semantics-preserving
(Made by human or LLM)

How to check that the edit preserves the semantics?

Example

```
def _retry(self, request, reason, spider):
    retries = request.meta.get('retry_times', 0) + 1
-   retry_times = self.max_retry_times
-   if 'max_retry_times' in request.meta: retry_times = request.meta['max_retry_times']
+   retry_times = request.meta.get('max_retry_times') or self.max_retry_times
    stats = spider.crawler.stats
    if retries <= retry_times:
        # (19 more, unchanged lines)
```

Commit message:

“Simplify retry_times assignment statement”

https://github.com/scrapy/scrapy/commit/694c6d3d (original edit)
https://github.com/scrapy/scrapy/commit/49c5afc5 (corrected edit)

Example

```
def _retry(self, request, reason, spider):
    retries = request.meta.get('retry_times', 0) + 1
-   retry_times = self.max_retry_times
-   if 'max_retry_times' in request.meta: retry_times = request.meta['max_retry_times']
+   retry_times = request.meta.get('max_retry_times') or self.max_retry_times
    stats = spider.crawler.stats
    if retries <= retry_times:
        # (19 more, unchanged lines)
```

Behavior changes if

`request.meta.get("max_retry_times")`

returns 0

Commit message:

“Simplify retry_times assignment statement”

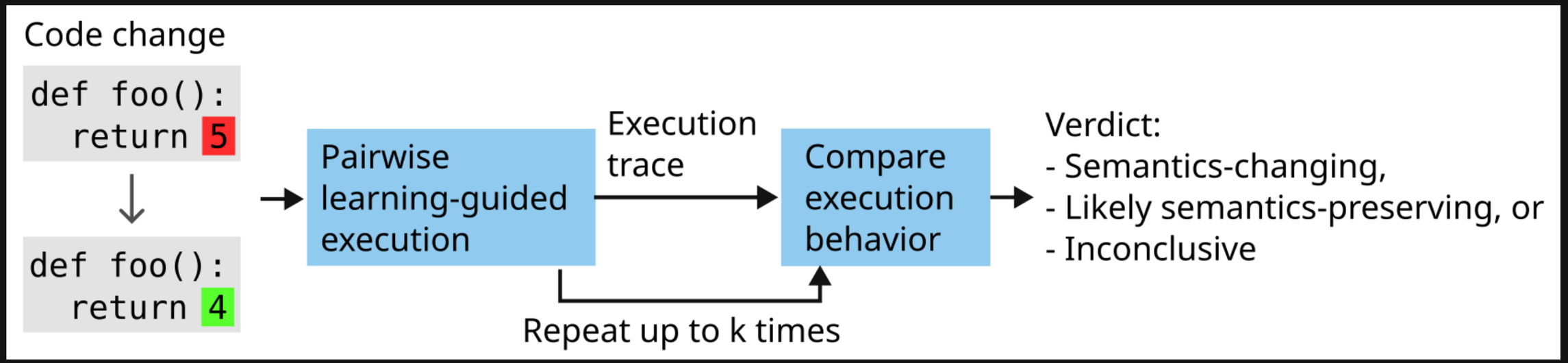
https://github.com/scrapy/scrapy/commit/694c6d3d (original edit)
https://github.com/scrapy/scrapy/commit/49c5afc5 (corrected edit)

How to Find Semantics-Breaking Changes?

Idea:

- **Execute** changed code **in isolation**
- **Compare** behavior: Old vs. new code

Overview of ChangeGuard



Learning-Guided Execution

Background:

“LExecutor: Learning-Guided Execution”

ACM SIGSOFT Distinguished Paper Award at FSE’23

- **Execute arbitrary code snippets**
- **Any missing values?**
  - Predict with a neural model and
  - inject into the execution

Example of Incomplete Code

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]

# ...
```

Example of Incomplete Code

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]

# ...
```

Example of Incomplete Code

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]


# ...
```

Example of Incomplete Code

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

Example of Incomplete Code

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

Example of LExecution

Let’s “lexecute” the 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]

# ...
```

Example of LExecution

Let’s “lexecute” the 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]

# ...
```

Example of LExecution

Let’s “lexecute” the code:

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 of LExecution

Let's "lexecute" the code:

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 of LExecution

Let's "lexecute" the code:

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

Pairwise Learning-Guided Execution

- Prior work: Single code snippets
- Here: Execute two similar code snippets side-by-side

Merge Snippets into Comparison Program

```
def f_old():
    # code of old code snippet

def f_new():
    # code of new code snippet

f_old()
f_new()
# Omitted: Code to compare behavior
```

Ensuring Consistency & Non-Interference

Want:
Both functions operate on same values

Want:
One function doesn't affect the other function

Solution: Consistency map

- Key: Access path (e.g., variable name) that refers to a predicted value
- Value: Pair of old and new value (deep copies)

Predicting Missing Values

- **Prior work (LExecutor): 23 hard-coded values**
- **Want: Diverse, project-specific, and realistic values**
- **Step 1: Neural model predicts one out of 12 abstract values ($\approx$ types)**
- **Step 2: Create concrete values from**
  - Literals found in code
  - Randomized generators of complex objects
  - Rich default objects that implements Python's special methods

Comparing Execution Behavior

- **Repeat pairwise learning-guided execution ($k = 300$ times)**
- **After the execution of old and new code:**
  - Compare argument and return values
  - Compare output written to console
  - Compare called functions
  - Compare raised exceptions

Classification of Code Changes

Three possible outcomes:

- Any difference in observed behavior:
  Semantics-changing
- If changed lines not reached:
  Inconclusive
- Otherwise:
  Likely semantics-preserving

Experimental Setup

- **Dataset 1: 224 hand-annotated commits**
  - Single-function changes from 10 popular projects
  - Commit message (does not) include(s) “refactor”, “simplify”, etc.
- **Dataset 2: 165 rule-based refactorings**[1]
- **Dataset 3: 445 LLM-generated code changes**
  - Prompt: “Improve code quality while preserving the behavior”

[1] RIdiom: Making Python Code Idiomatic by Automatic Refactoring Non-Idiomatic Python Code with Pythonic Idioms (FSE’22)

Effectiveness (Dataset 1)

Ground truth		ChangeGuard		
	Total	Changing	Preserving	Inconclusive
Changing	131	91	12	28
Preserving	93	27	48	18
Total	224	118	60	46

Precision: 77.1%

Recall: 69.5%

Accuracy when giving an answer: 78.1%

Effectiveness (Datasets 2 & 3)

Refactoring tool	Prediction		
	Inconclusive	Changing	Preserving
RIdiom	38	5	122
GPT-3.5	31	87	69
GPT-4	38	143	77

Effectiveness (Datasets 2 & 3)

Refactoring tool	Prediction		
	Inconclusive	Changing	Preserving
RIdiom	38	5	122
GPT-3.5	31	87	69
GPT-4	38	143	77

Caused by a bug in RIdiom

Effectiveness (Datasets 2 & 3)

Refactoring tool	Prediction		
	Inconclusive	Changing	Preserving
RIdiom	38	5	122
GPT-3.5	31	87	69
GPT-4	38	143	77

Caused by a bug in RIdiom

Manually inspected 30+30:
21+16 indeed change the semantics

Examples

“Refactored” by GPT-3.5:

```
- if isinstance(arr_or_dtype, ExtensionType): return arr_or_dtype.name == "category"
- if arr_or_dtype is None: return False
- return CategoricalDtype.is_dtype(arr_or_dtype)
+ if isinstance(arr_or_dtype, ExtensionType) and arr_or_dtype.name == "category": return True
+ elif arr_or_dtype is None: return False
+ else: return CategoricalDtype.is_dtype(arr_or_dtype)
```

Examples

“Refactored” by GPT-3.5:

```
- if isinstance(arr_or_dtype, ExtensionType): return arr_or_dtype.name == "category"
- if arr_or_dtype is None: return False
- return CategoricalDtype.is_dtype(arr_or_dtype)
+ if isinstance(arr_or_dtype, ExtensionType) and arr_or_dtype.name == "category": return True
+ elif arr_or_dtype is None: return False
+ else: return CategoricalDtype.is_dtype(arr_or_dtype)
```

Not semantically equivalent

Examples

“Refactored” by GPT-4:

```
    def param_allowed(stat_name, include, exclude):
      if not include and not exclude: return True
-     for p in exclude:
-         if p in stat_name: return False
-     if exclude and not include: return True
-     for p in include:
-         if p in stat_name: return True
-     return False
+     if any(p in stat_name for p in exclude): return False
+     if include: return any(p in stat_name for p in include)
+     return not exclude
```

Examples

Behavior changes when empty, empty, non-empty

"Refactored" by GPT-4:

```
    def param_allowed(stat_name, include, exclude):
      if not include and not exclude: return True
-     for p in exclude:
-         if p in stat_name: return False
-     if exclude and not include: return True
-     for p in include:
-         if p in stat_name: return True
-     return False
+     if any(p in stat_name for p in exclude): return False
+     if include: return any(p in stat_name for p in include)
+     return not exclude
```

Don’t Regression Tests Find This?

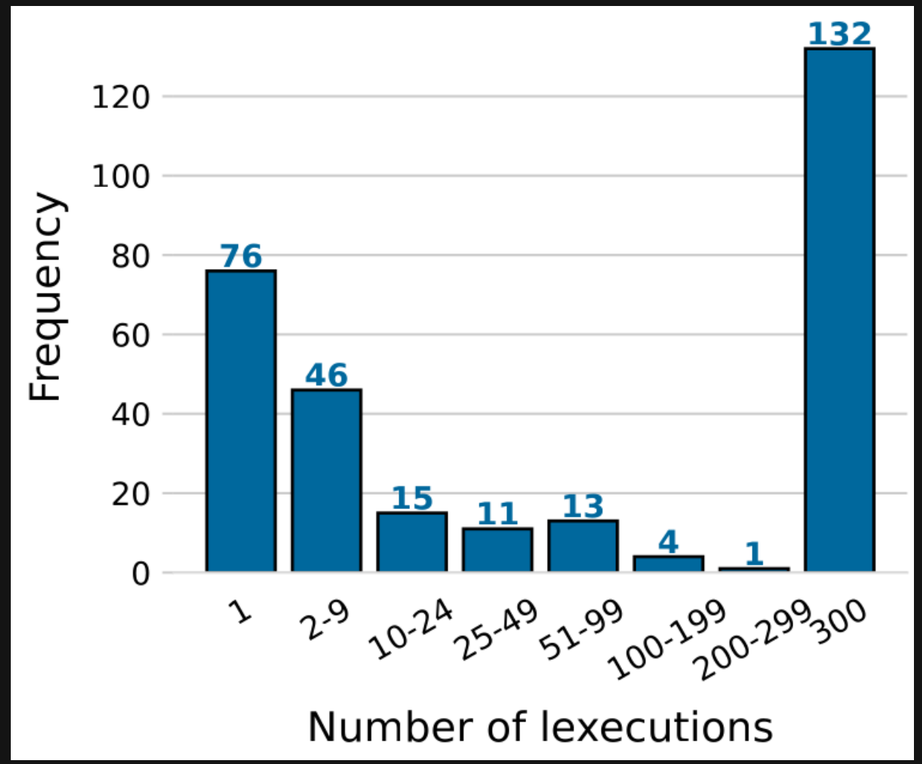
Ground truth		ChangeGuard			Regression testing		
	Total	Changing	Preserving	Inconclusive	Changing	Preserving	Inconclusive
Changing	131	91	12	28	10	29	92
Preserving	93	27	48	18	2	18	73
Total	224	118	60	46	12	47	165

Precision: 83%
Recall: 8%
Accuracy: 48%

No tests available or project fails to build

Efficiency

- First execution: 2.82 seconds
- Additional executions: 1.01 seconds each
- Nb. executions to produce a verdict:



Conclusion

- **ChangeGuard**: New approach for identifying semantics-breaking changes
- Conceptual novelty: **Pairwise, learning-guided execution**
- Usage scenarios: Validate **human-made** or **LLM-proposed refactorings**

ChangeGuard: Validating Code Changes via Pairwise Learning-Guided Execution
https://arxiv.org/abs/2410.16092