CS527 midterm report

Yiteng Hu*
Yuehao Shi*
yitengh2@illinois.edu
yuehaos2@illinois.edu
UIUC
Champaign, IL, USA

ABSTRACT

The article presents an extension of the FreeFuzz automated fuzz testing tool to test PaddlePaddle, an emerging open-source deep learning library. The team aims to improve the reliability of deep learning libraries by identifying and fixing potential bugs and vulnerabilities. The proposed solution involves four steps: code collection, instrumentation, mutation test, and oracle test. The team has completed the code collection and instrumentation stages, while the mutation stage is yet to be completed. The team has encountered challenges due to different installation settings and environments for TensorFlow and PaddlePaddle packages and insufficient data collection, delaying the mutation strategy development. The team plans to use metrics such as covered APIs, the size of the value space, and line coverage to evaluate the effectiveness of the Freefuzz tests for PaddlePaddle and compare them to other state-of-the-art deep learning library testing techniques.

KEYWORDS

Fuzz testing, Deep learning libraries, PaddlePaddle, Automated testing, Mutation testing

ACM Reference Format:
Yiteng Hu and Yuehao Shi. 2018. CS527 midterm report. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 PROBLEM

Deep learning (DL) has become an indispensable tool for solving complex problems in various domains, including computer vision, natural language processing, and speech recognition. DL libraries, such as TensorFlow and PyTorch, are essential tools for building, optimizing, and running DL models. However, testing these libraries can be challenging due to their complexity and the lack of effective testing techniques.

Testing DL systems is challenging because they typically comprise multiple layers of neural networks that perform intricate computations on large amounts of data. These computations involve non-linear transformations that are often difficult to reason about. Additionally, DL systems are commonly trained on vast datasets that are challenging to reproduce, adding to the complexity of testing.

To address these challenges, researchers have proposed various techniques for testing DL systems, including mutation testing, adversarial testing, differential testing, and fuzzing. Fuzzing is a testing technique that involves generating random inputs to test software systems. It has been successfully applied to various types of software systems, including web applications, operating systems, and compilers.

FreeFuzz is a recent approach that utilizes open source mining to fuzz DL libraries. This automated fuzz testing tool has been effectively employed to test TensorFlow and PyTorch libraries by generating a large number of test cases from code snippets in library documentation, developer tests, and DL models in the wild. FreeFuzz uses a line coverage metric to measure its code coverage.

However, testing the emerging open-source deep learning library, PaddlePaddle, using automated techniques like FreeFuzz presents challenges due to the complexity of its public APIs, which are mainly exposed in Python. The dynamic typing used in these APIs makes it challenging to automatically determine the API input parameter types. Furthermore, PaddlePaddle library has unique features and functionalities that require specific testing techniques, making it necessary to extend FreeFuzz to test this library.

The project team aims to extend the existing fuzzing system to test PaddlePaddle library. The goal is to improve the reliability of these libraries by discovering and fixing potential bugs and vulnerabilities. The significance of this research lies in the fact that DL libraries play a crucial role in our daily lives, but bugs in these systems can have disastrous consequences. Surprisingly, despite the importance of DL library reliability, there is only limited work for testing DL libraries. One example of this is CRADLE, which uses pre-existing DL models to test Keras and its backends, and addresses the issue of test oracles through differential testing. To expand on this approach, LEMON enhances CRADLE by utilizing various model mutation rules to create a wider range of DL models. This enables the invocation of more library code and exposes a greater number of potential bugs in the DL library.

In conclusion, while DL systems are complex and challenging to test using conventional techniques, fuzzing has emerged as a promising approach to testing DL libraries. FreeFuzz has been effectively used to test TensorFlow and PyTorch libraries, and extending it to test PaddlePaddle library will further enhance the reliability of

*Both authors contributed equally to this research.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

DL libraries. This research contributes to improving the quality of DL libraries and reducing the likelihood of disastrous consequences caused by bugs in these systems.

2 SOLUTION

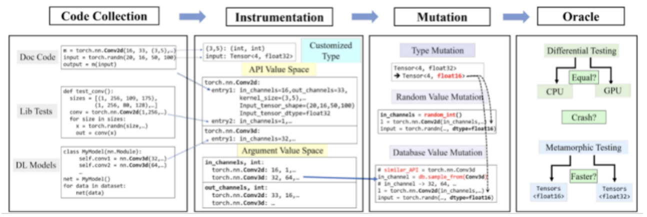
To address the challenge of testing PaddlePaddle library using automated techniques like FreeFuzz, we propose extending FreeFuzz to test PaddlePaddle library using the same method mentioned in the FreeFuzz paper. More specifically, we will obtain code/models from three different sources: 1) code snippets from the library documentation, 2) library developer tests, and 3) DL models in the wild. By combining these three sources of input, we can generate a large number of test cases that cover different parts of the PaddlePaddle library.

To address the challenge of dynamic typing in Python, we will use type annotations to specify the input parameter types for each API. Type annotations provide a way to specify the expected types of function arguments and return values in Python [15]. By using type annotations, we can automatically determine the API input parameter types and generate valid test cases.

We will use the same line coverage metric used in the FreeFuzz paper to measure our code coverage. This metric measures the percentage of lines of code executed during testing. We will also evaluate our approach on a set of benchmark programs that use PaddlePaddle library as their backend engine.

In conclusion, extending FreeFuzz to test PaddlePaddle library is an important step towards improving the quality and reliability of DL libraries. By using the same method mentioned in the FreeFuzz paper and incorporating type annotations for dynamic typing in Python, we can generate effective tests for PaddlePaddle library and improve its robustness and correctness. Furthermore, our approach can help identify and fix bugs in PaddlePaddle library, which can lead to better performance and accuracy of DL models built using this library. Our approach can also be extended to test other DL libraries that are mainly exposed in Python and have dynamic typing.

Overall, our proposed approach can significantly improve the testing of DL libraries and help ensure their correctness and reliability. By generating a large number of test cases that cover different parts of the library, we can increase the code coverage and identify potential bugs or issues. This can ultimately lead to better quality DL models and more reliable applications built using these models.



```
m = torch.nn.Conv2d(16, 33, (3,5),…)
input = torch.randn(20, 16, 50, 100)
output = m(input)
```

```
def test_conv():
  sizes = [(1, 256, 109, 175),
           (1, 256, 80, 128),…]
  conv = torch.nn.Conv2d(1,256,…)
  for size in sizes:
    x = torch.randn(size,…)
    out = conv(x)
```

```
class MyModel(nn.Module):
    self.conv1 = nn.Conv3d(32,…)
    self.conv2 = nn.Conv3d(64,…)
    …
net = MyModel()
for data in dataset:
    net(data)
```

```
(3,5): (int, int)
input: Tensor<4, float32>
```

```
torch.nn.Conv2d:
  entry1: in_channels=16,out_channels=33,
          kernel_size=(3,5),…
          Input_tensor_shape=(20,16,50,100)
          Input_tensor_dtype=float32
  entry2: in_channels=1,…
torch.nn.Conv3d:
  entry1: in_channels=32,…
```

```
in_channels, int:
  torch.nn.Conv2d: 16, 1,…
  torch.nn.Conv3d: 32, 64,…
out_channels, int:
  torch.nn.Conv2d: 33, 16,…
  torch.nn.Conv3d: …
```

```
Tensor<4, float32>
→ Tensor<4, float16>
```

```
in_channels = random_int()
l = torch.nn.Conv2d(in_channels,…)
input = torch.randn(…, dtype=float16)
```

```
# similar_API = torch.nn.Conv3d
in_channel = db.sample_from(Conv3d)
# in_channel -> 32, 64, …
l = torch.nn.Conv2d(in_channels,…)
input = torch.randn(…, dtype=float16)
```

Figure 1: Work flow of Freefuzz

3 EVALUATION

The team will use a set of metrics to evaluate the effectiveness of Freefuzz tests for PaddlePaddle. These metrics include the number of covered APIs, the size of the value space, and line coverage. Additionally, we will compare the Freefuzz testing metrics with those of other state-of-the-art deep learning library testing techniques, such as LEMON and CRADLE. By assessing these metrics, the team hopes to determine the strengths and weaknesses of Freefuzz testing and how it compares to other approaches. This information will help us to improve their testing process and ensure the quality of PaddlePaddle. At present, the team has not amassed a sufficient quantity of API and value space data, precluding the provision of evaluation results up to this point. In light of this, the team is currently engaged in an exploration of the potential application of Lemon and Cradle in order to facilitate a comparison of the results of the final fuzzing tests.

Currently, the team has not collected enough API and value space data, which prevents them from providing evaluation results. Therefore, the team is currently investigating the use of testing tools such as Lemon and Cradle, which have adapted the PaddlePaddle library to compare the results of final fuzzing tests. In the next stage, the team plans to include the total number of bugs found in the evaluation results.

4 RESULT ANALYSIS

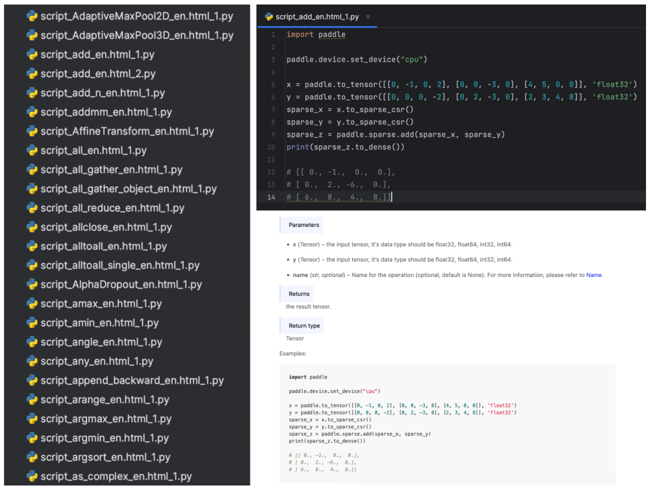
4.1 Code Collection

The methodology for collecting code snippets for the PaddlePaddle framework was inspired by the FreeFuzz research paper. The team obtained code snippets from three primary sources, namely: (1) code snippets from the official documentation, (2) testing files from the PaddlePaddle library developer tests, and (3) deep learning models in the wild. The team also collected API executions of PaddlePaddle based on the same three sources.

The official website of PaddlePaddle was the primary source of code snippets, which provided all available API names, definitions, and execution examples for the latest version of the framework. To automate the process of parsing the documentation and obtaining the code snippets, the team utilized the bs4 Python package. The resulting scripts and outputs for crawling can be accessed from the `/src/reptile directory`, while the API execution code snippets are available in `src/reptile/code_snippets`. After filtering out invalid results from the reptile tool and several API execution codes that caused instrumentation to crash, the team obtained a total of 826 API names, 720 API definitions, and 1493 executable code snippets from the official PaddlePaddle documentation website.

The team also collected testing files from the PaddlePaddle library developer tests. There are a total of 692 testing files in the latest version of PaddlePaddle, which the team obtained by cloning the latest PaddlePaddle official repository into their local machine.

Lastly, the team obtained code from deep learning models in the wild. The team collected publicly available PaddlePaddle models and utilized them as a source of real-world use cases for testing the PaddlePaddle libraries. For instance, the PaddleNLP package, which is an easy-to-use and powerful NLP library that leverages PaddlePaddle as a basic structure, provided numerous Paddle API calls that were suitable for collecting APIs for fuzzing. The team executed all testing files for this package to augment their API pool with this package.



```
script_AdaptiveMaxPool2D_en.html_1.py
script_AdaptiveMaxPool3D_en.html_1.py
script_add_en.html_1.py
script_add_en.html_2.py
script_add_n_en.html_1.py
script_addmm_en.html_1.py
script_AffineTransform_en.html_1.py
script_all_en.html_1.py
script_all_gather_en.html_1.py
script_all_gather_object_en.html_1.py
script_all_reduce_en.html_1.py
script_allclose_en.html_1.py
script_alltoall_en.html_1.py
script_alltoall_single_en.html_1.py
script_AlphaDropout_en.html_1.py
script_amax_en.html_1.py
script_amin_en.html_1.py
script_angle_en.html_1.py
script_any_en.html_1.py
script_append_backward_en.html_1.py
script_arange_en.html_1.py
script_argmax_en.html_1.py
script_argmin_en.html_1.py
script_argsort_en.html_1.py
script_as_complex_en.html_1.py
```

script_add_en.html_1.py

```python
import paddle

paddle.device.set_device("cpu")

x = paddle.to_tensor([[0, -1, 0, 2], [0, 0, -3, 0], [4, 5, 0, 0]], 'float32')
y = paddle.to_tensor([[0, 0, 0, -2], [0, 2, -3, 0], [2, 3, 4, 8]], 'float32')
sparse_x = x.to_sparse_csr()
sparse_y = y.to_sparse_csr()
sparse_z = paddle.sparse.add(sparse_x, sparse_y)
print(sparse_z.to_dense())

# [[ 0., -1.,  0.,  0.],
# [ 0.,  2., -6.,  0.],
# [ 6.,  8.,  4.,  8.]]
```

Parameters

- **x** (*Tensor*) – the input tensor, it's data type should be float32, float64, int32, int64.
- **y** (*Tensor*) – the input tensor, it's data type should be float32, float64, int32, int64.
- **name** (*str, optional*) – Name for the operation (optional, default is None). For more information, please refer to Name.

Returns

the result tensor.

Return type

Tensor

Examples:

```python
import paddle

paddle.device.set_device("cpu")

x = paddle.to_tensor([[0, -1, 0, 2], [0, 0, -3, 0], [4, 5, 0, 0]], 'float32')
y = paddle.to_tensor([[0, 0, 0, -2], [0, 2, -3, 0], [2, 3, 4, 8]], 'float32')
sparse_x = x.to_sparse_csr()
sparse_y = y.to_sparse_csr()
sparse_z = paddle.sparse.add(sparse_x, sparse_y)
print(sparse_z.to_dense())

# [[ 0., -1.,  0.,  0.],
# [ 0.,  2., -6.,  0.],
# [ 6.,  8.,  4.,  8.]]
```

Figure 2: Code snippets list (left), code snippet detail (right top), and code snippet origin (right bottom)

4.2 instrumentation

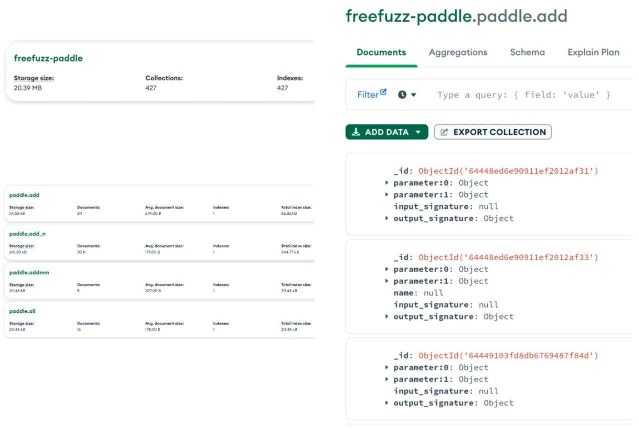
The second stage of the Freefuzz adaptation process for PaddlePaddle involves dynamic tracing with instrumentation, which allows Freefuzz to intercept selected PaddlePaddle APIs and collect information about their execution, including input argument values and output values. This information is stored in MongoDB in JSON format to create the necessary type space, API value space, and argument value space for later fuzzing stages.

To implement this instrumentation stage, the team repurposed the code used for testing the PyTorch library, utilizing functions such as `hijack`, `decorate_class`, `decorate_function`, and `write_fn`. Different sources of code collection require different ways of implementing instrumentation. During the process, the team found that not all collected APIs are valid for instrumentation, and they decided to focus on fuzzing Paddle functions, keeping 511 APIs over 720 defined APIs on the Paddle official website.

For the code collected from the official documentation, the team executed all 1493 code snippets, but several code snippets could not compile during instrumentation due to typo errors on the Paddle official documentation examples. After executing all code snippets and hijacking API execution information into MongoDB, there are a total of 420 collections in MongoDB after running all code snippets, where one collection means the execution information of one API. Over 511 selected Paddle APIs, 420 APIs get executed and stored, so official documentation code snippets contributed 82% of targeted API.

For the official test code, the team cloned Paddle source code from Github and executed all `test_*.py` files in Paddle/test folder using pytest as automation testing tool. After removing unsupported imports by removing invalid tests, there are 2741 tests left. After executing all 2741 tests, only one more collection shown on MongoDB, but the total size in database increased 2%, indicating significant API execution overlap between official documentation code snippets and testing documents.

For open-source projects, the team chose the PaddleNLP package and ran testing files inside it. After executing all testing files in PaddleNLP, there are 6 more collections, and 22% more MB data increased in MongoDB, meaning that PaddleNLP contributes additional API execution data in total data. PaddleNLP focuses on a specific area of APIs and generates more in-depth API calls compared to Paddle official tests, which have a large proportion of unexecutable tests and false testing files that hamper their contribution to the total data.



freefuzz-paddle

Storage size:	Collections:	Indexes:
20.39 MB	427	427

	Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
paddle.add	24.58 kB	211	274.00 B	1	36.86 kB
paddle.add_n	610.30 kB	30 K	179.00 B	1	544.77 kB
paddle.addmm	20.48 kB	5	327.00 B	1	20.48 kB
paddle.all	20.48 kB	16	178.00 B	1	20.48 kB

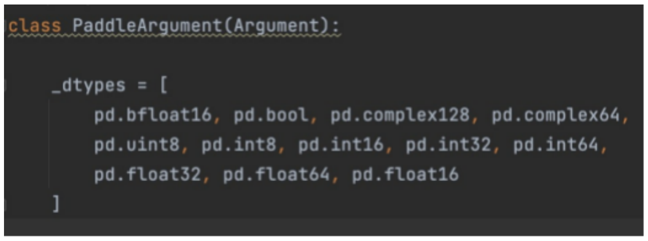
freefuzz-paddle.paddle.add

```
_id: ObjectId('64448ed6e90911ef2012af31')
▸ parameter:0: Object
▸ parameter:1: Object
  input_signature: null
▸ output_signature: Object

_id: ObjectId('64448ed6e90911ef2012af33')
▸ parameter:0: Object
▸ parameter:1: Object
  name: null
  input_signature: null
▸ output_signature: Object

_id: ObjectId('64449103fd8db6769487f84d')
▸ parameter:0: Object
▸ parameter:1: Object
  input_signature: null
▸ output_signature: Object
```

Figure 3: Database overview (left top), Database collections (left bottom), and Collection detail (right)

4.3 Mutation Test

We collected different types for PaddlePaddle, which are shown in the figure3. And we decide to extend the mutation strategies used in the original paper. To summarize, the team has selected a subset of APIs as a proof of concept to demonstrate the feasibility of applying the complete Freefuzz testing process to the PaddlePaddle package, as set out in the midterm goals.



```python
class PaddleArgument(Argument):

    _dtypes = [
        pd.bfloat16, pd.bool, pd.complex128, pd.complex64,
        pd.uint8, pd.int8, pd.int16, pd.int32, pd.int64,
        pd.float32, pd.float64, pd.float16
    ]
```

Figure 4: data types in PaddlePaddle

4.4 Test Oracle

5 CONCLUSIONS AND FUTURE WORK

In general, the team has made good progress towards achieving the midterm goals except for the mutation strategy script in stage 3. However, the team encountered some challenges that slowed down the development process. One of the team members was using a MacOS device with an M1 chip, which required different installation settings and environments for TensorFlow and PaddlePaddle packages compared to general MacOS installation instructions. This unexpected issue took some time for the team to resolve. Additionally, developing a mutation strategy was challenging without sufficient data collection from stage 1. As a result, the team has decided to postpone the development of this part until stage 1 is almost complete.

As previously mentioned, the team used a limited number of PaddlePaddle APIs in the midterm to initiate the project. To increase the sample size, the team is currently developing Python crawler scripts to collect additional APIs from official documentation, test documents, and open source projects.

The third stage of Freefuzz involves mutation-based fuzzing, where FreeFuzz will generate mutants for the test inputs collected from stage 2. At this point, the team has not developed a mutation strategy but plans to implement type mutation, random value mutation, and database value mutation in the next phase of work. The fourth stage entails running all the generated tests with oracles. Currently, with the collected unmutated API values, there have been no crash or runtime error outputs, which is as expected. In the next phase, the team plans to run the mutated code and generate more oracles to try to find bugs in the PaddlePaddle packages.

Finally, the team will evaluate the effectiveness of the Freefuzz testing methodology by measuring the number of covered APIs, the size of the value space, and line coverage. Additionally, they will compare the Freefuzz testing metrics with those of LEMON and CRADLE. With the successful setup of the development environment and a deeper understanding of Freefuzz, the team is confident that they can complete one functionality in each paragraph mentioned above each week before the final deadline and complete the project on time.

6 GITHUB LINK

https://github.com/yuehaoshi/FreeFuzz

REFERENCES

[1] Wei, A., Deng, Y., Yang, C., Zhang, L. (2022, May). Free lunch for testing: Fuzzing deep-learning libraries from open source. In Proceedings of the 44th International Conference on Software Engineering (pp. 995-1007).