SOAR: Synthesis for Open-Source API Refactoring

Aidan Yang. Advisors: Ruben Martins, Claire Le Goues

Queen's University

Carnegie Mellon University

Introduction

Motivation:
- Developers working on data science applications have an abundance of APIs and API versions to choose.
- The manual refactoring between data-science APIs is a tedious and error-prone task.
- Developers need to manually read documentation to understand hyperparameter mappings.

Our Tool:
- Instead of using a large program corpus for statistical learning [1], SOAR uses readily available and highly maintained open-source API documentation to learn API representation mappings.
- SOAR uses documentation descriptions and compilation error messages to produce SMT constraints for the Z3 SMT solver [2].

```
1  import tensorflow.keras.layers as tf
2  ....
41
42  self.conv1 = tf.Conv2D(                          A
                    filters=32,
                    kernel_size=3,
                    strides=(2, 2))
43  ....                                             B
48  self.dense1 = tf.Dense(10)
49  self.dense2 = tf.Dense(1568)
50  ....                                             C
55  self.deconv1 = tf.Conv2DTranspose(
                    filters=64,
                    kernel_size=3,
                    strides=2)
56  ....                                             D
63  self.relu6 = tf.ReLU()
```

```
1  import torch
2  ....
39
40  self.var0 = torch.nn.Conv2d(
                    in_channels=32,
                    out_channels=32,
                    kernel_size=(3, 3),
                    stride=(2, 2),
                    padding=(0, 0))
41  ....
51  self.var5 = torch.nn.Linear(64 , 10)
52  self.var6 = torch.nn.Linear(10, 1568)
53  ....
57  self.var8 = lambda t: t.permute(0, 3, 1, 2)
58  self.var9 = torch.nn.ConvTranspose2d(
                    in_channels=32,
                    out_channels=64,
                    kernel_size=(3, 3),
                    stride=(2, 2),
                    padding=(0, 0))
59  ....
62  self.var13 = torch.nn.ReLU()
```

Figure 1. Example refactoring between Tensorflow and Pytorch

Approach

1. Collect documentation artifacts from official DL API sites [3, 4].

2. Create benchmarks based on open-source github repos API usage and documentation type description.

3. Match with target API.

4. Synthesize target API program.

5. Generate constraints based on error messages.

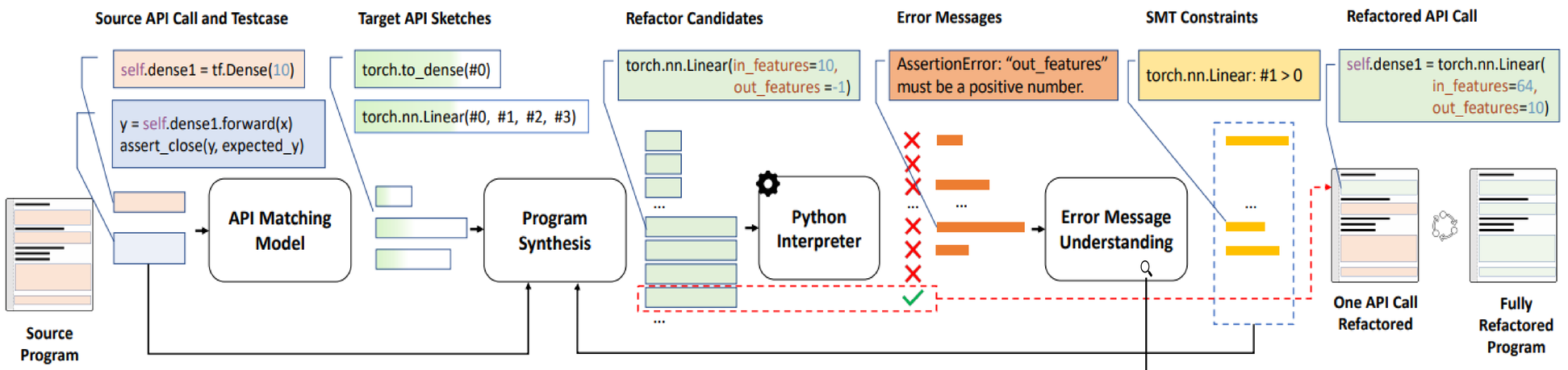

Figure 2. Overview of SOAR's architecture

Results

- Our automatic test generation model creates 20 benchmarks for Tensorflow to PyTorch, and 20 benchmarks from R Dplyr to Python Pandas. We run 5266 total evaluations.
- Mean run-time of 97.23±141.58 seconds, and a median of 14.76 seconds.
- On average, SOAR refactored an input of 13.6 ± 12.14 lines of code to an output of 18.56 ± 16.40 lines of code, through one-to-many mapping.

	SOAR	SOAR w/o Specs.	SOAR w/o Err. Msg.
conv_pool_softmax(4L)	**1.60**	23.02	14.35
img_classifier(8L)	**12.82**	336.00	65.66
three_linear(3L)	3.18	**2.34**	21.07
embed_conv1d_linear(5L)	**5.27**	123.85	16.90
word_autoencoder(3L)	1.81	**1.46**	2.64
gan_discriminator(8L)	**12.80**	timeout	252.20
two_conv(4L)	16.69	timeout	**15.09**
img_autoencoder(11L)	**160.97**	391.09	487.54
alexnet(20L)	425.22	timeout	**66.13**
gan_generator(9L)	**412.47**	timeout	timeout
lenet(13L)	**280.91**	timeout	timeout
tutorial(10L)	**6.04**	timeout	58.29
conv_for_text(11L)	**9.04**	timeout	32.29
vgg11(28L)	**40.83**	timeout	132.67
vgg16(38L)	**82.05**	timeout	139.27
vgg19(44L)	**83.99**	timeout	189.90

Figure 4. Execution time for 16 benchmarks

Specs: SMT constraint generated by specifications of available API documentation.
Err. Msg: SMT constraints generated by run-time error messages.

Error message constraint generation:

POS tag each error message word and begin constraint generation model
1. Customize 4 POS tag patterns as hyponyms [5] (e.g., NN IN JJ NN: tensor with negative dimension)
2. Use hyponym to find candidate faulty parameter and its constraint
3. Mutate program with changed parameter.

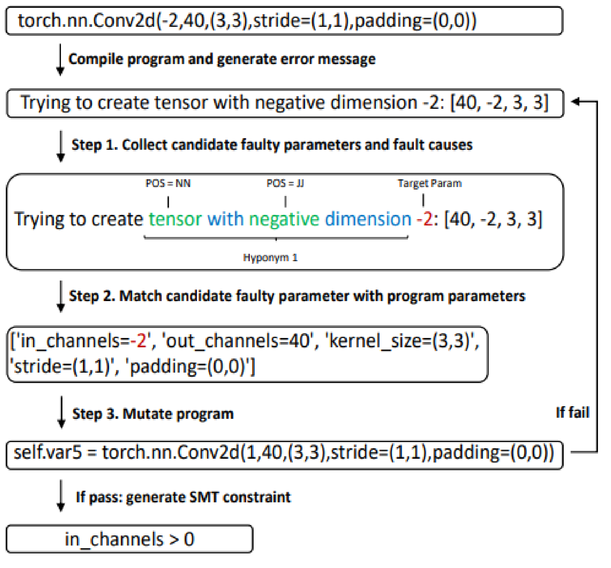

Figure 3. Example constraint generation

References

[1] A.T.Nguyen, H.A.Nguyen,T.T.Nguyen,and T.N.Nguyen. Statistical learning approach for mining api usage mappings for code migration. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 457–468, 2014.
[2] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
[3] Api documentation : Tensorflow core v2.2.0. https://www.tensorflow. org/api docs/index.html, july 2020.
[4] Pytorch documentation. https://pytorch.org/docs/stable/index.html, july 2020.
[5] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Coling 1992 volume 2: The 15th international conference on computational linguistics*, 1992