SOAR: Synthesis for Open-Source API Refactoring

Aidan Z.H. Yang

Queen's University

Carnegie Mellon University

Motivation

- As software evolves, the API calls it depends on may become obsolete [1].

- No existing work that takes an enumerative program synthesis approach towards API refactoring.

- Existing work learns and recommends potential API mappings [1,2], or migrates APIs based on a large dataset of existing migrations [3].

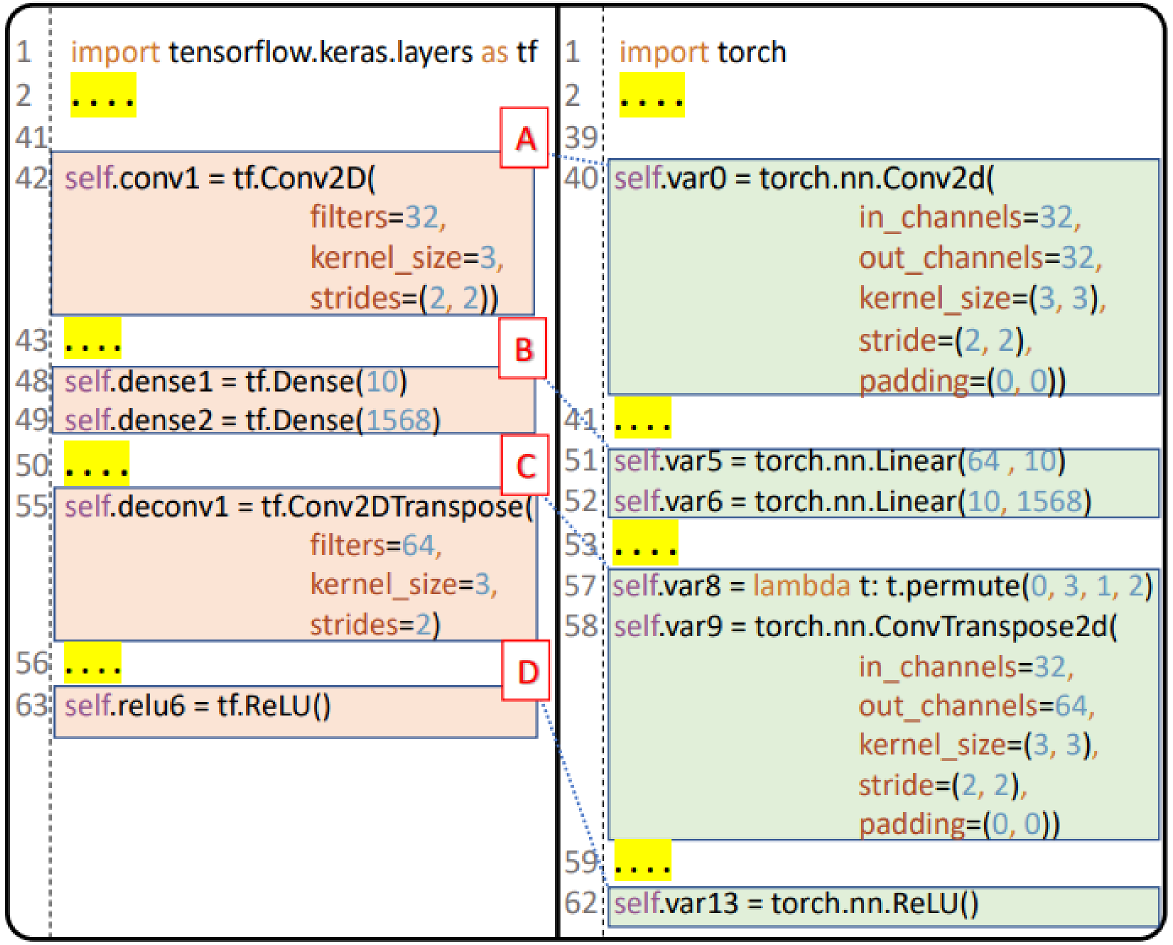
[1] Perkins, Jeff H. "Automatically generating refactorings to support API evolution." *Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. 2005.
[2] Bui, Nghi. "Towards zero knowledge learning for cross language API mappings." *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019.
[3] Bui, Nghi DQ, Yijun Yu, and Lingxiao Jiang. "SAR: learning cross-language API mappings with little knowledge." *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019.
[4] Phan, Hung Dang, et al. "Statistical migration of API usages." *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017.

Motivation

A: tf.Conv2d parameter **filters** has no corresponding name parameter.

B: **tf.Dense** has no torch function of the same name.

C: Two functions from torch are required to perform the same task as **tf.Conv2DTranspose.**



Key Insights

Documentation on data science APIs have description on parameters, as well as call examples.

A `tf.Tensor` has the following properties:

- a data type (float32, int32, or string, for example)
- a shape

Each element in the Tensor has the same data type, and the data type is always known.

In eager execution, which is the default mode in TensorFlow, results are calculated immediately.

```
>>> # Compute some values using a Tensor
>>> c = tf.constant([[1.0, 2.0], [3.0, 4.0]])
>>> d = tf.constant([[1.0, 1.0], [0.0, 1.0]])
>>> e = tf.matmul(c, d)
>>> print(e)
tf.Tensor(
[[1. 3.]
 [3. 7.]], shape=(2, 2), dtype=float32)
```

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string, optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool, optional*) – If `True`, adds a learnable bias to the output. Default: `True`

Key Insights

Error messages contain information on the input matrix.

- Trying to create tensor with negative dimension -2: [40, -2, 3, 3].
- embedding(): argument weight (position 1) must be Tensor, not int.
- Expected 3-dimensional input for 3- dimensional weight [2, 2, 3], but got 4- dimensional input of size [100, 50, 40, 1] instead.

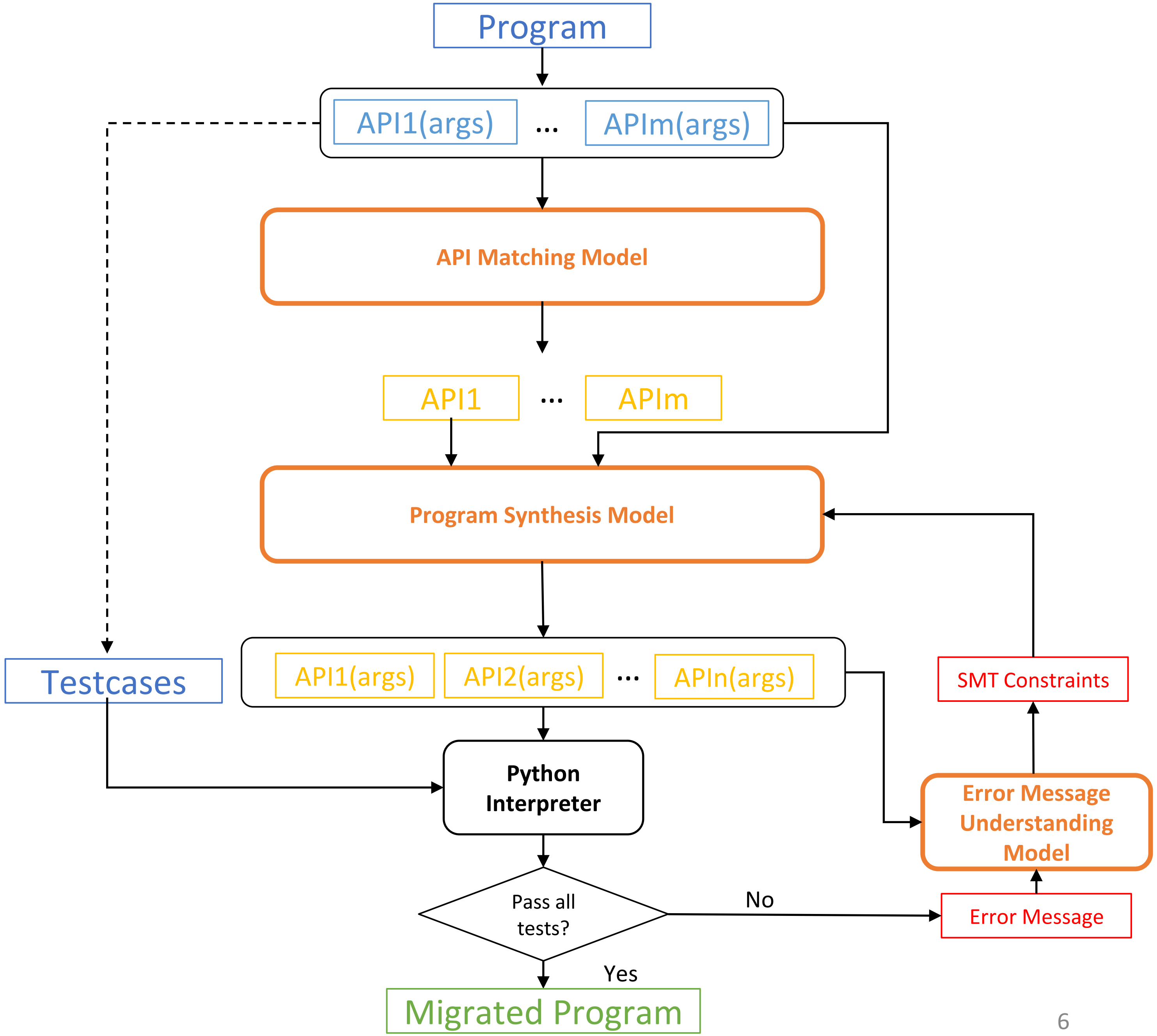
General Framework

Input:

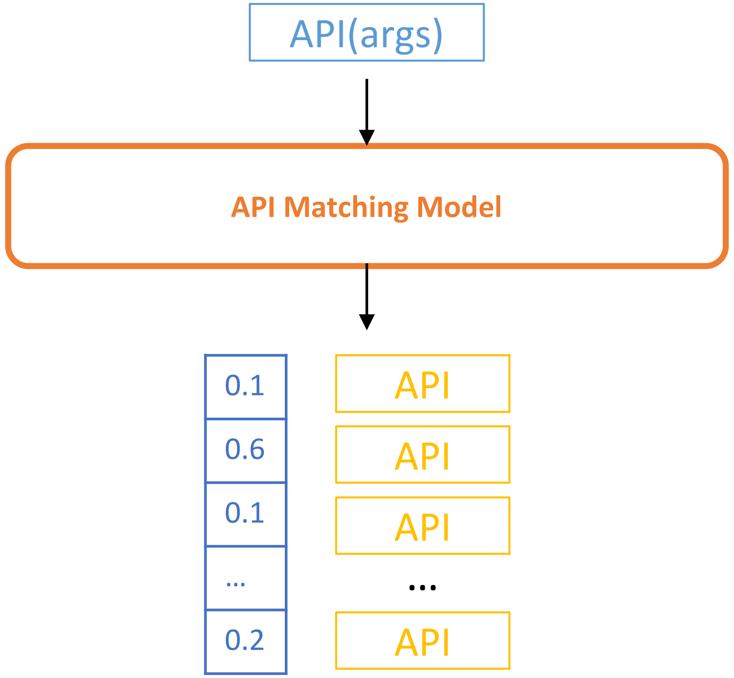
- A program written with source API.
- Documentation of APIs.
- Some test cases for the original program.

Output:

- Program with the same functionality written with the target library.



API Matching



Output:

- A probability distribution over all target APIs

  -> Top 10 most similar APIs by cosine-similarity.

Model:

- API representation learning, represent each API in a continuous vector space.
- Learned with the API documentation (page title, code example etc…) and GloVe pre-trained NLP data.

$$\text{Embedding}(\mathbf{x^i}) = \sum_{i=j}^{n} \frac{x_j^i \cdot \mathbf{w_j}}{\sum_{t=0}^{m} x_j^t}$$

$$\text{sim}(\text{Rep}(\mathbf{x}^i), \text{Rep}(\mathbf{x}^j)) = \frac{\text{Rep}(\mathbf{x}^i) \cdot \text{Rep}(\mathbf{x}^j)}{|\text{Rep}(\mathbf{x}^i)||\text{Rep}(\mathbf{x}^j)|}$$

Program Synthesis

Algorithm 1 SYNTHESIZER($\mathcal{I}, \mathcal{S}, \mathcal{T}, \mathcal{C}$)

Input: $\mathcal{I}$: existing program, $\mathcal{S}$: source library, $\mathcal{T}$: target library, $\mathcal{C}$: test cases
Output: $\mathcal{O}$: refactored program

$\vec{r}$: API mapping = MAPAPI($\mathcal{T}, \mathcal{S}$)
$\mathcal{O} = \{\}$
for each $l \in \mathcal{I}$ **do**
    $\mathcal{O} = \mathcal{O} \cup$ REFACTORLINE($l, \mathcal{T}, \mathcal{C}, \vec{r}$)
end for

Algorithm 2 REFACTORLINE($l, \mathcal{T}, \mathcal{C}, \vec{r}$)

Input: l: line of code from $\mathcal{I}$, $\mathcal{T}$: target library, $\mathcal{C}$: test cases, $\vec{r}$: ranked list of API matchings
Output: $\mathcal{R}$: refactored snippet

$\mathcal{O} = \{\}$
for each $l' \in \vec{r}$ **do**
    $\vec{s}$ = GENERATESKETCHES($l', \mathcal{T}$)
    for each $s \in \vec{s}$ **do**
        $\mathcal{R}$ = FILLSKETCH(s)
        if PASSTESTS($\mathcal{R}, \mathcal{C}$) **then**
            return $\mathcal{R}$
        end if
    end for
end for

Program Sketching:
Sketch a new program with holes based on the original program template.

Program Enumeration
Enumerate through all likely target calls with all likely parameters.

Pruning the Search Space

Synthesizer before and after the SMT constraints

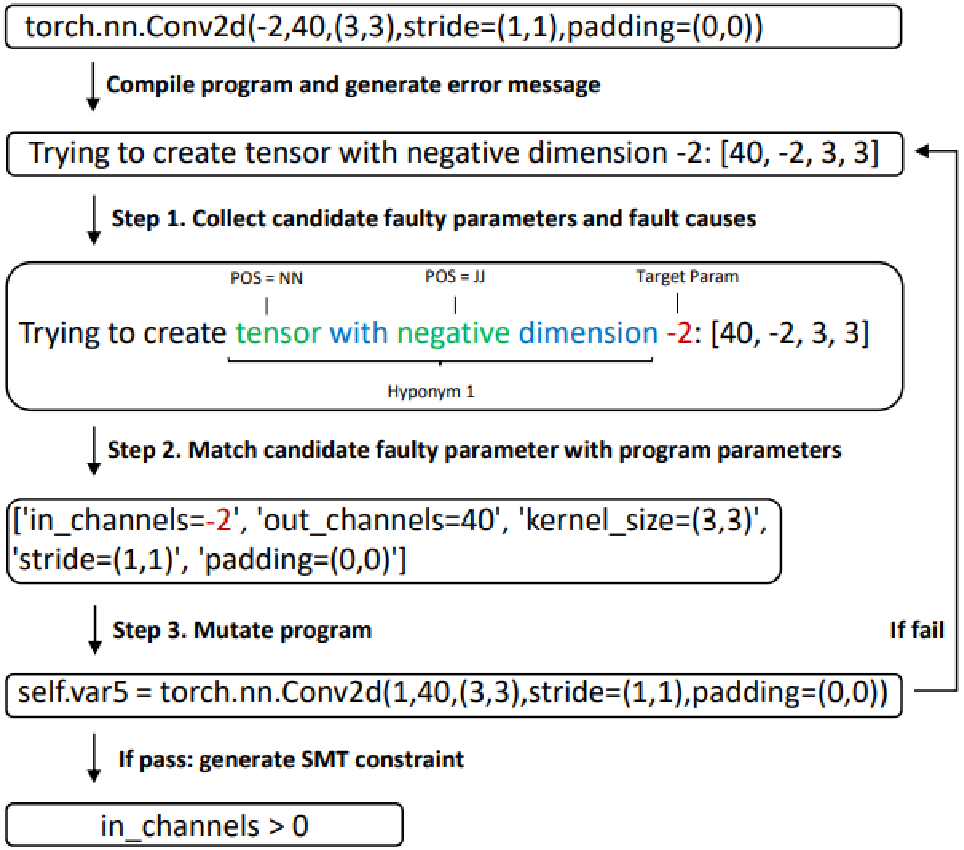
1. **Specification inferred from documentation.**

   - **in_channels** (*int*) – Number of channels in the input image

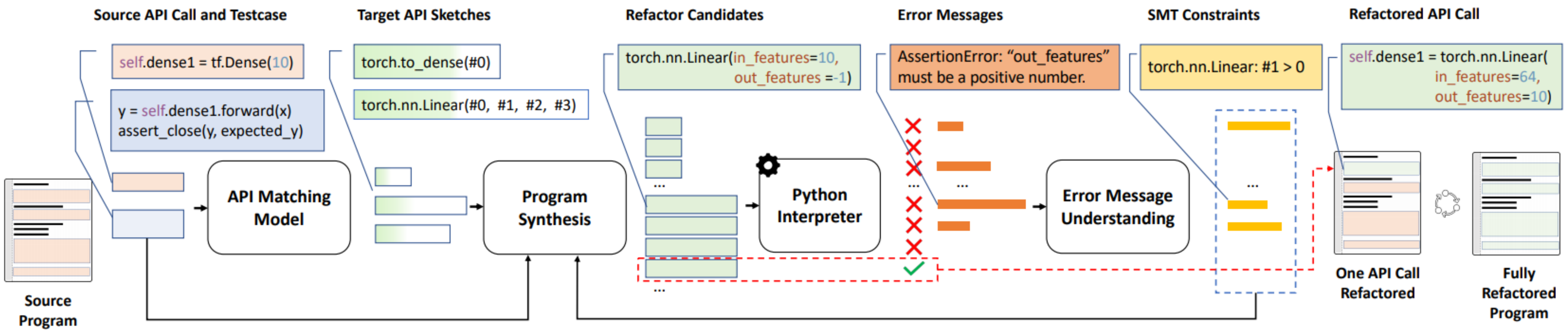
2. **Constraints from Error messages.**

Error Message Understanding

- Generate SMT constraints from run-time errors.

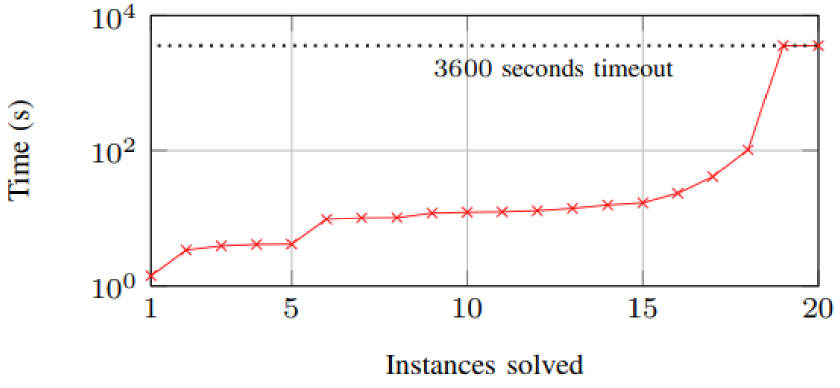


Example Pipeline



Results

	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
densenet_main1(5L)	timeout	timeout	timeout
densenet_main2(3L)	timeout	timeout	timeout
densenet_conv_block(6L)	timeout	timeout	timeout
densenet_trans_block(3L)	timeout	timeout	timeout



- Collected the top **1015** starred repositories that have TensorFlow as a topic tag.
- Over 8 million lines of code and over 500K TensorFlow API calls.
- **76%** of the repositories use API calls included in our benchmarks at least once.

Conclusion

- SOAR is the first to use NLP and synthesis for automatic API refactoring.
- SOAR requires no existing migration as training data.
- SOAR can successfully migrate 80% of neural network programs composed by 3 to 44 layers in with an average time of 97.23 seconds.
- For Dplyr to Pandas migration, 90% of benchmarks are solved on average in 17.31 seconds.
- With ablation studies, we show that the use of specifications from API documents and learning from error messages can improve synthesis performance.

Hire Me ☺

- I am applying for PhD positions for **Fall 2021!**

- a.yang@queensu.ca
- **https://aidanby.github.io/**
- @AidanYang5