Motivation - Documentation, which provides a high-level description of the task performed by the source code, is a must-have for team-based software development groups. Conversely, even with various tools that have been developed to aid the programmer during its creation, crafting correct and consistent documentation for source codes remains a labor-intensive task. Thus, significant gaps in documentation are ubiquitous and considerably contribute to the inefficiency of program comprehension by software engineers.

Goal – generate high-level natural language descriptions from raw low-leve Python source code.

What we built - our final product is a 178M parameter Transformer with pre-trained embeddings that takes as input a function's source code and predicts its description.

Dataset: For our dataset we use 108,726 Python function-description pairs taken from Wan et al.’s paper [9].

Preprocessing: We split the dataset into 80:10:10 for train, dev, and test sets, respectively; assign the examples into buckets based on the source code length; split data points into batches, tokenize them and apply padding based on buckets.

Model

- We use a Transformer[8] with pre-trained GloVe[5], and GPT-2[6] embeddings for the inputs and outputs, respectively.
- Multi-Head attention – masking, combined with output embeddings offset by one position, to ensure predictions would only depend on the known outputs (the previous output tokens).

\[
Attention(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V
\]

- Embeddings – For our encoder, we pre-trained a GloVe language model on Python code for functions and lines. For our decoder, we used pre-trained embeddings from Hugging Face’s implementation of GPT-2 small [10].

<table>
<thead>
<tr>
<th>Epochs</th>
<th>30</th>
<th>N (encoder-decoder layers)</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_model</td>
<td>768</td>
<td>Multi-Head Attention Layers</td>
<td>8</td>
</tr>
<tr>
<td># of parameters</td>
<td>178 M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Discussion

- There is a considerable variance problem in the model. The fact that the dev set loss distances itself so early from the training set loss indicates that the model might be overtly big for our dataset.
- The striking disconnect between the sets seems to indicate that there was a representation issue in our algorithm. We suspect the GloVe embeddings were insufficient in capturing the structural essence of the code. This dissipated most of the meaning through that layer and hurt the generalizability of the model.
- A more interesting approach would be to use abstract syntax trees, such as in code2vec[2], with some form of tree encoder which would capture more of the structural nature of source code.

Results

- Loss function - we use negative log likelihood (NLL) with a log-softmax activation applied on their linear output matrix.

\[
\text{loss}(x, t) = -\log \left( \frac{e^x}{\sum_j e^{x_j}} \right)
\]

<table>
<thead>
<tr>
<th>Training Loss</th>
<th>Dev Loss</th>
<th>Loss (10 epochs)</th>
</tr>
</thead>
</table>