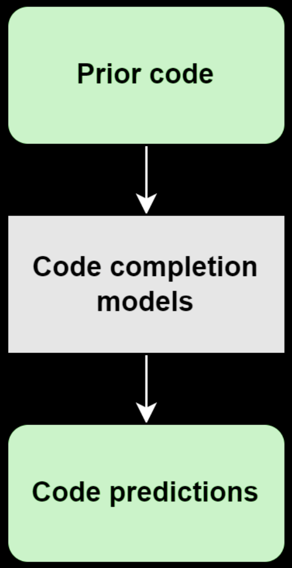
Boosting Performance of Code Completion Algorithms with Abstract Syntax Trees and Convolutional Graph Autoencoders

CS5814 Final project

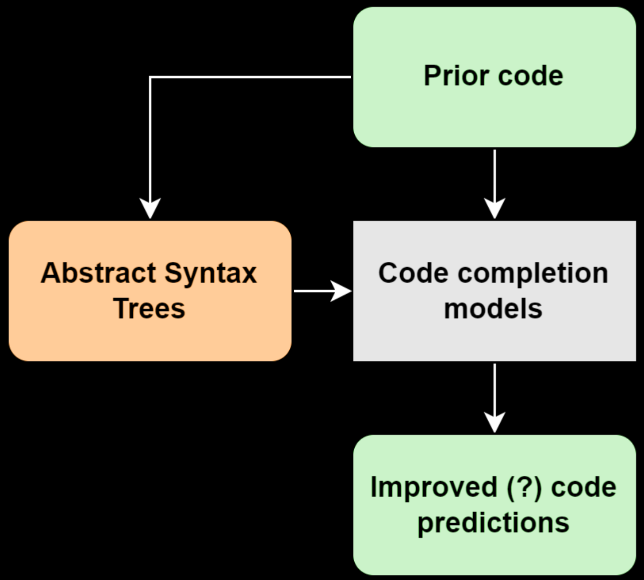
Sam Donald

Problem Statement

- Standard code completion models rely solely on preceding code.



- Yet additional (free) information is available in the form of Abstract Syntax Trees.

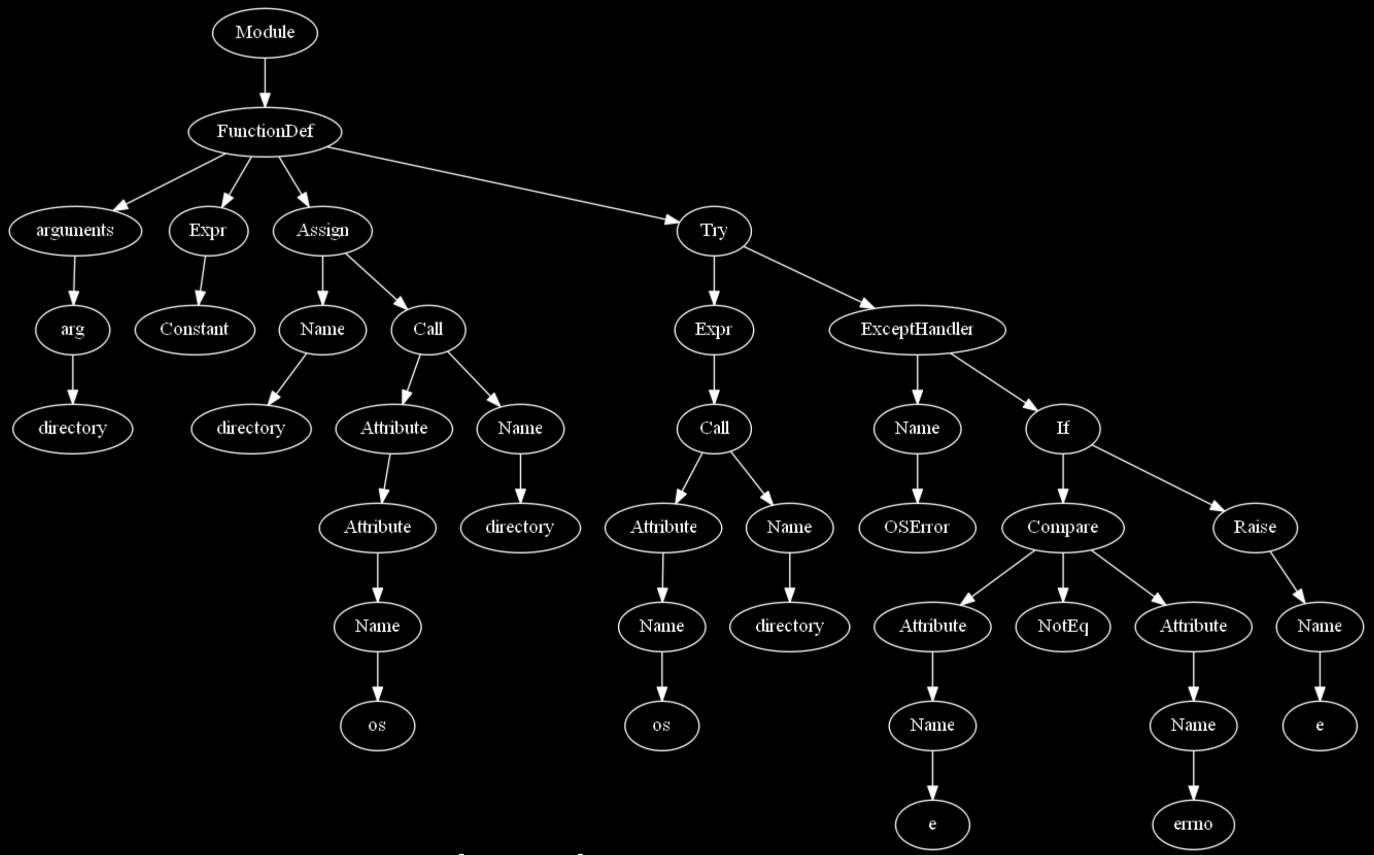


- How can this free information be integrated into code completion models?

Abstract Syntax Trees (ASTs)

- ASTs provide a graph representation of the syntactic structure of code
- Typically used by compilers
- Can only be generated from complete and valid functions

```
def ensure_directory(directory):
    # Create the directories along the provided directory path that do not exist.
    directory = os.path.expanduser(directory)
    try:
        os.makedirs(directory)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise e
```



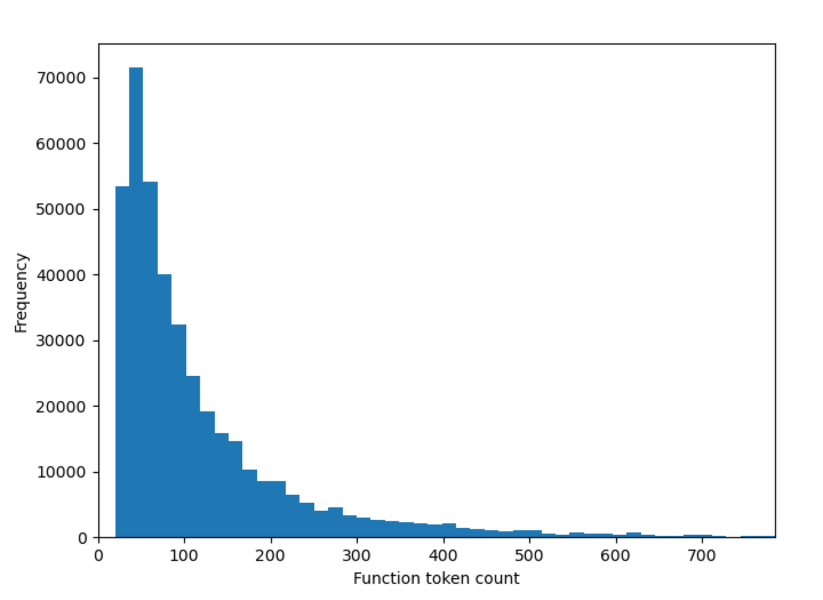
Input function

Rendered AST

Data Description

- The python portion of the CodeSearchNet dataset is used for code prediction and AST generation
- The following information is provided for ~457,000 functions sourced from GitHub

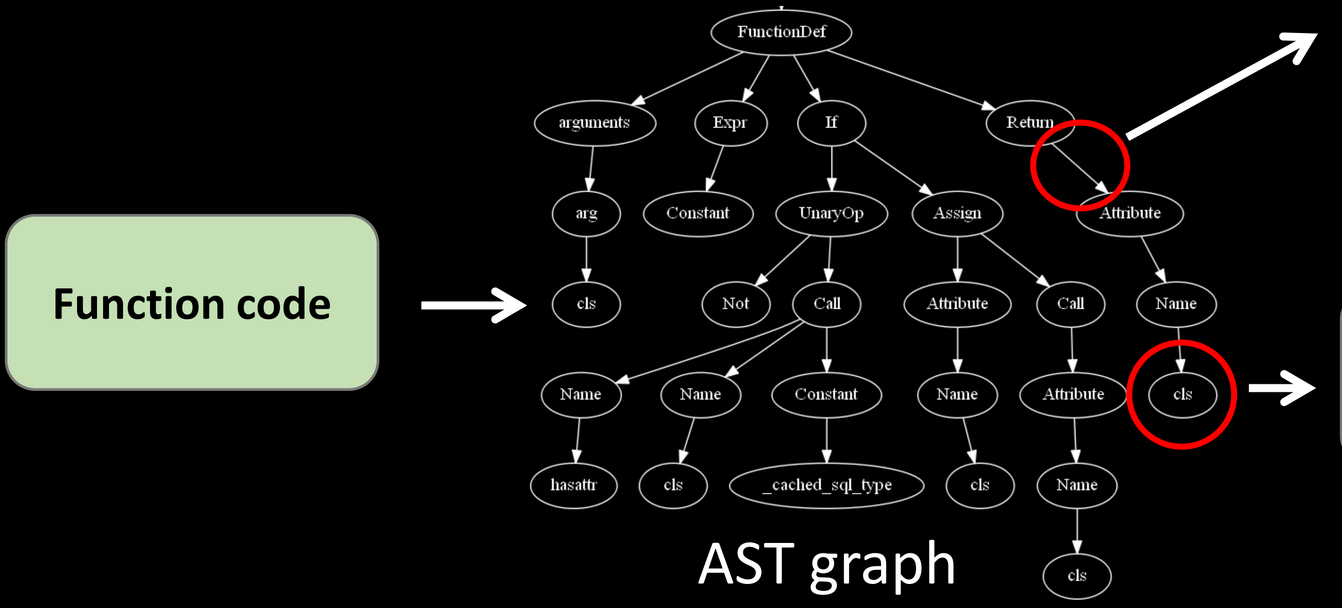
    - Repository name
    - Function path in repository
    - Function name
    - Entire function string
    - Language
    - Function code URL
    - Function documentation string
    - Function documentation tokens
    - <u>**Function code string**</u> **– used to generate ASTs**
    - <u>**Function code tokens**</u> **– used as inputs for code prediction**

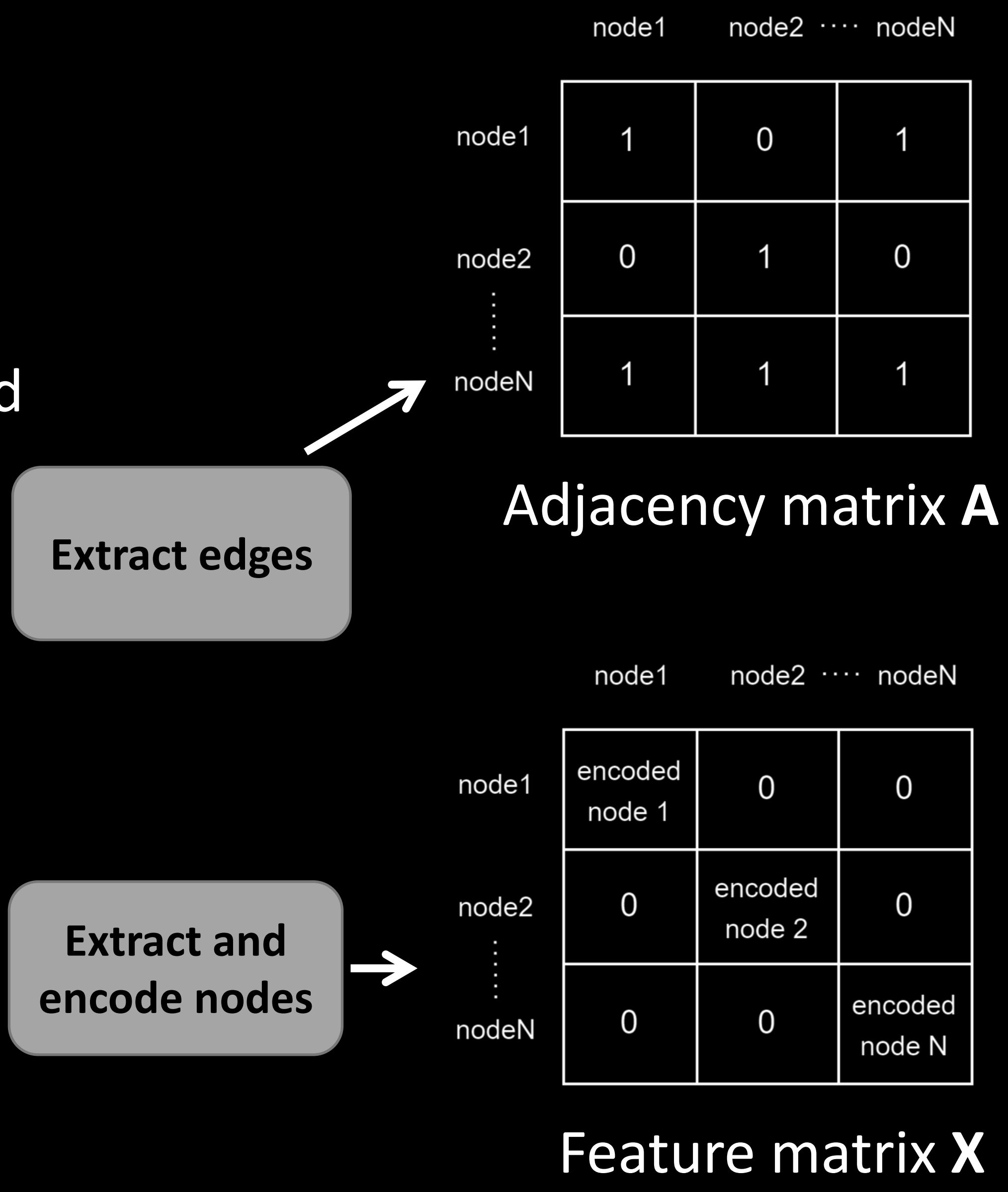


Function token count distribution for CodeSearchNet test set

Data Preprocessing (AST)

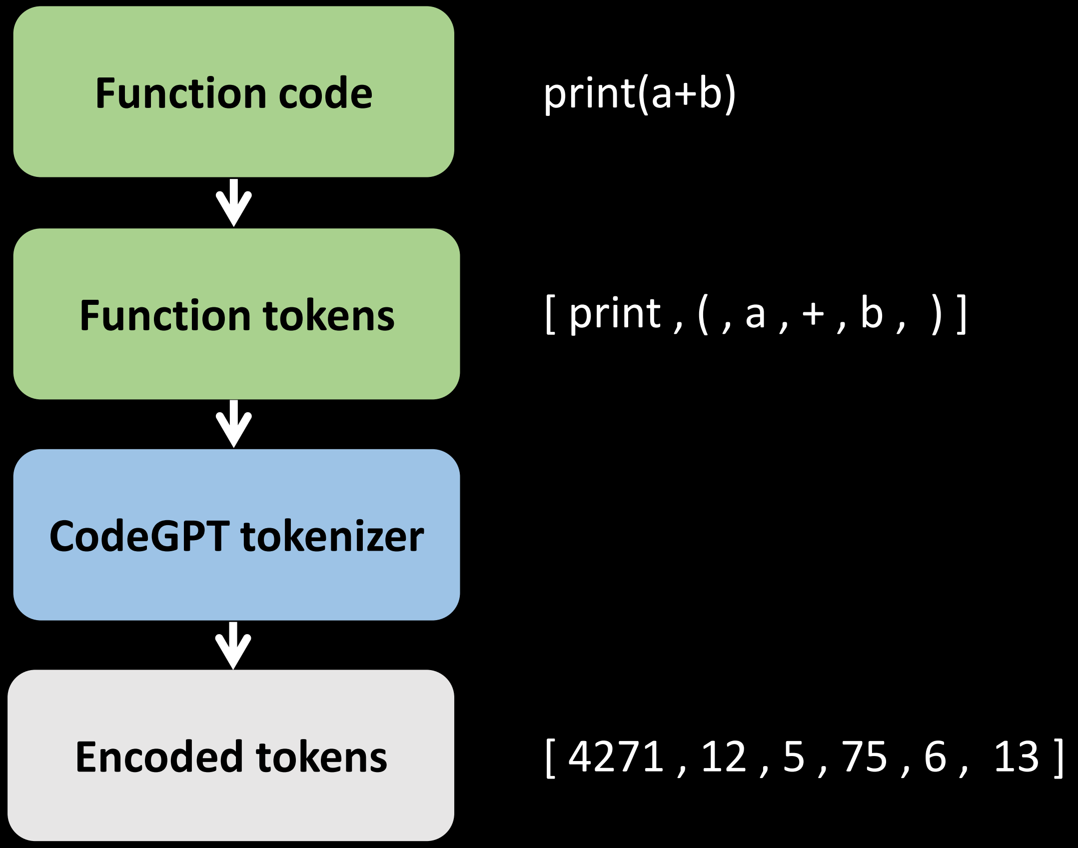
- A generated AST graph is walked to extract all edges and node names
- 10,000 most frequent names are one hot encoded
- Remainder are grouped to a shared encoding





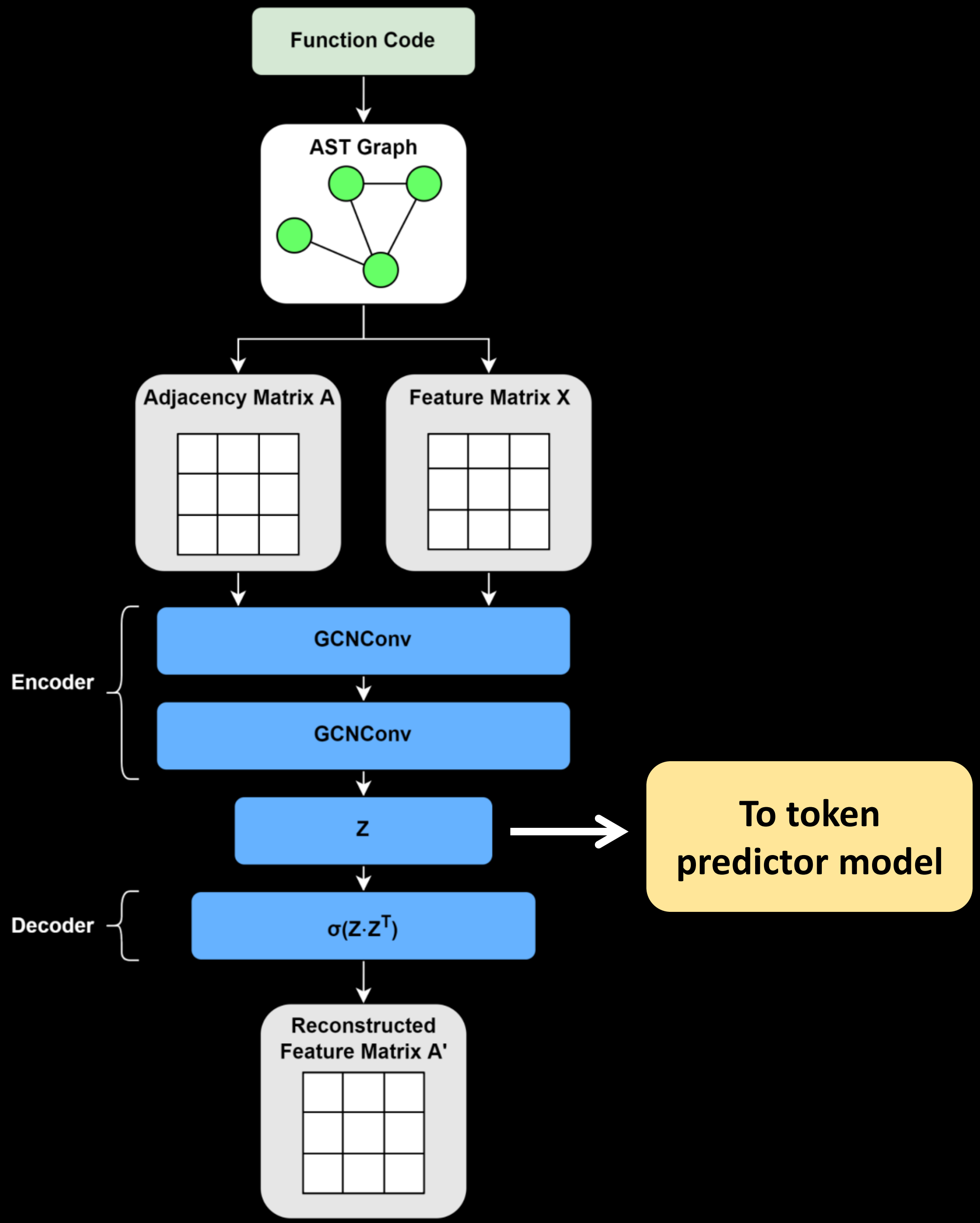
Data Preprocessing (Tokens)

- Function tokens are extracted from the CodeSearchNet data and passed to a CodeGPT tokenizer
- The tokenizer is pretrained based on a python dataset
- Comments are removed prior to encoding via the GPT tokenizer



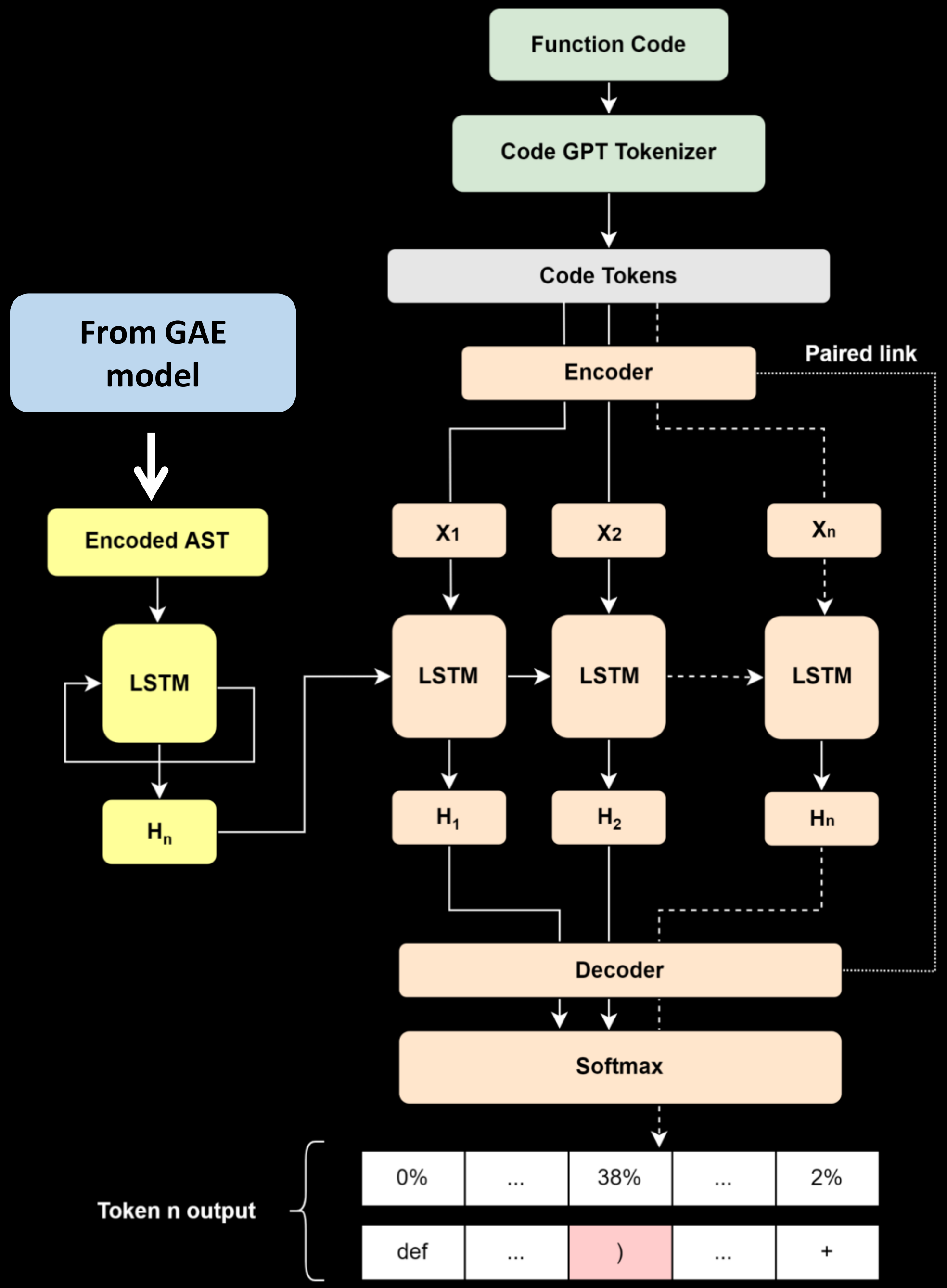
GAE Model

- A Graph Autoencoder (GAE) is used to encode the AST graph structure into then latent variable **Z**
- The latent variable is then decoded to reconstruct the feature matrix **A**
- The encoder portion of the GAE consists of 2 Graph Convolutional (GCNConv) layers
- The latent variable **Z** is then fed to the token prediction model



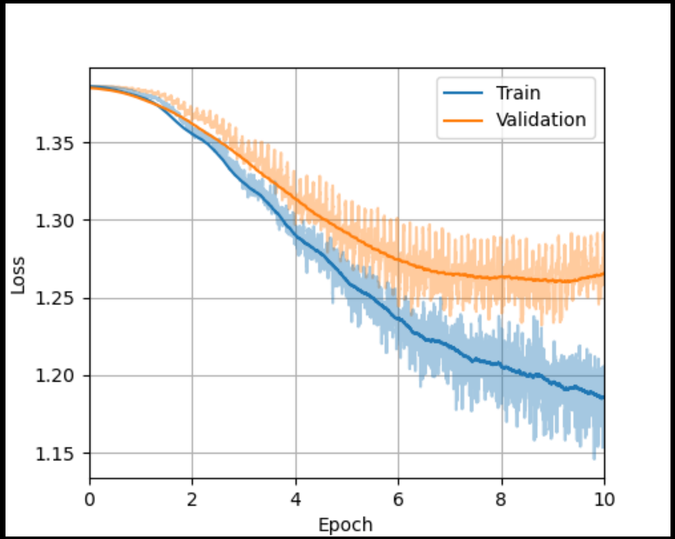
Token prediction model

- An LSTM based encoder-decoder model is used for next token predictions
- The variable sized latent variable **Z** from the GAE is transformed via a LSTM (yellow) into a constant sized hidden variable
- This hidden variable is used to initialize the encoder-decoder LSTM (orange)
- The encoder-decoder layers can be linked, such that their weights are shared and easier to learn during training.



Results

Graph Autoencoder

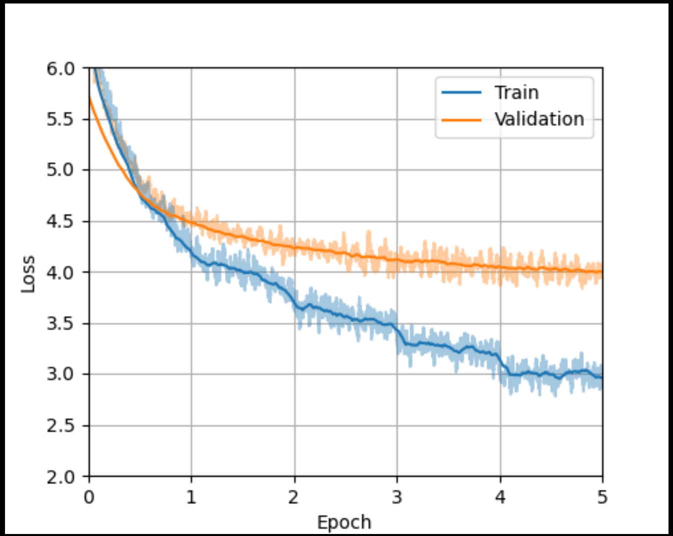


Training loss

		Truth	
		+	-
Prediction	+	0.84	0.54
	-	0.16	0.46

Normalized confusion matrix for reconstructed graph edges

Code Prediction



Training loss (with encoded AST)

	Accuracy
No encoded AST	34.87%
Encoded AST	**35.62%**

Test accuracy

Conclusions

- While the Graph Autoencoder was able to capture a portion of the AST structure, it performed poorly overall as evident by the large false positive rate
  - This is likely due to the sparsity of the node adjacency matrix **A.**
  - Preprocessing steps such as graph pruning would reduce sparsity and likely improve the ability for the GAE to encode the AST.
- The inclusion of the encoded AST information boosted token prediction accuracy by 0.75%
  - However, as these ASTs are generated from the entire function, it is possible the model is using this information to “look into the future” and access future tokens.
  - Further work is needed to validate these results, by constructing partial ASTs based only on code preceding the token of interest.

Lessons Learned

- Graph Autoencoders are difficult to train with sparse data.
- Developing a unique neural network architecture is also difficult, and more time should be allocated for its implementation and debugging.
- When results are poor, attention should be placed into understanding the underling reasons, instead of trying in vain to improve results.
- Working by yourself on large projects, without regular feedback from others, can result in the exploring of unnecessary rabbit holes without a clear direction.
- The space of potential neural network models is huge and very exciting!