BigDat 2020
Ancona, Italy

BIG CODE

(Part 2/3)

Dr. Jayanti Prasad

6th Winter school on Big Data

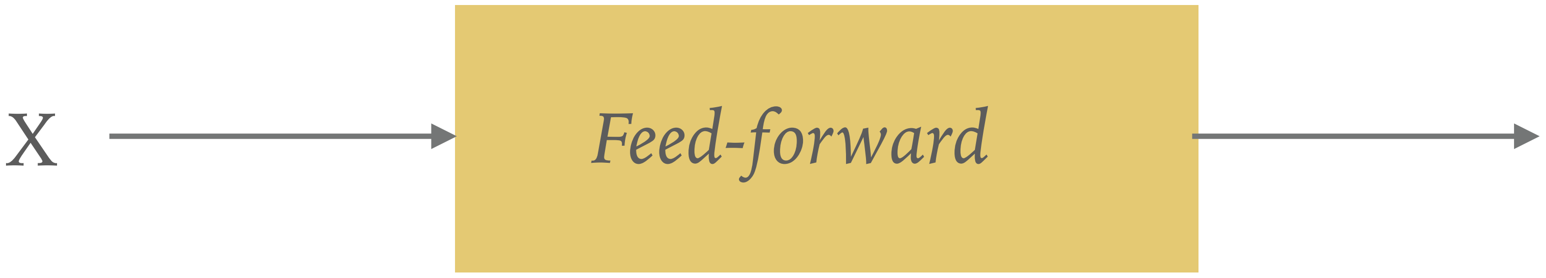
(13-18 January, 2020)

Ancona, ITALY

PLAN

- LSTM Networks
- Abstract Syntax Tree (AST)
- Bblfish parser
- Github data crawling
- Data Preparation
- Model building
- Seq2Seq model
- TreeLSTM
- Discussion and conclusions

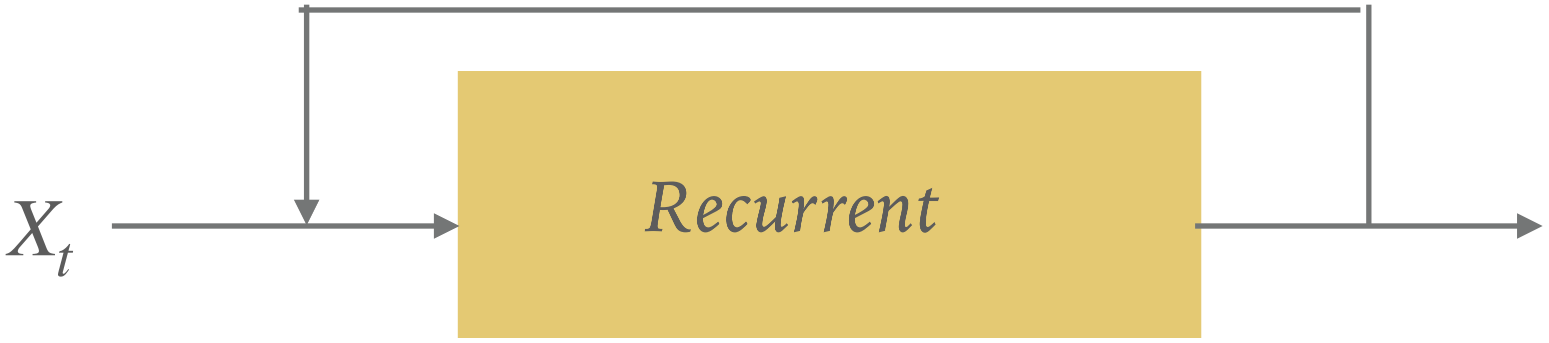
NEURAL NETWORKS



X

Feed-forward

$$Y = f(X) = \sigma\left(\sum_i w_i X_i + b_i\right)$$



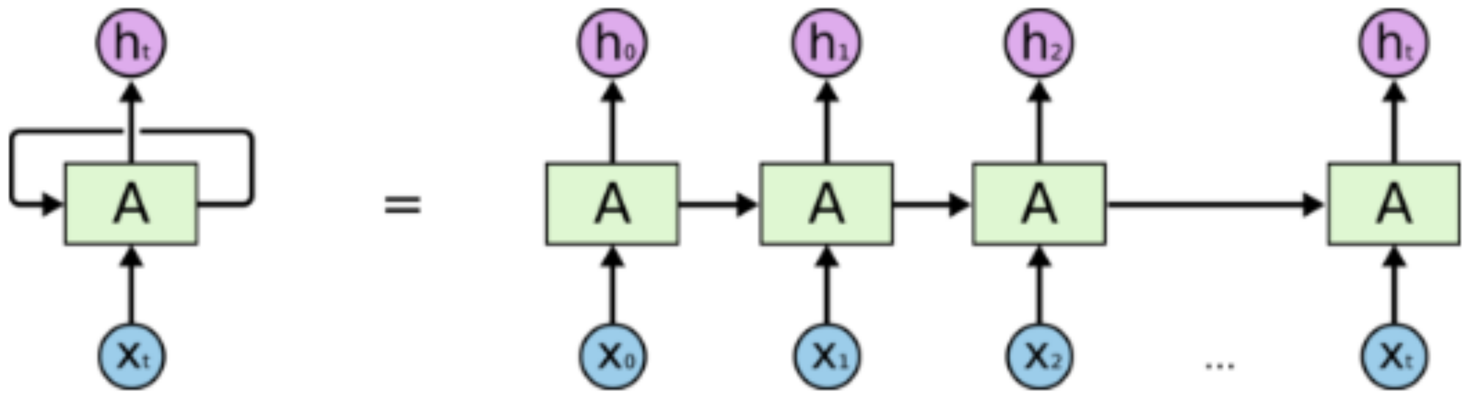
X_t

Recurrent

$$h_t = f(X_t, h_{t-1})$$
$$y_t = g(h_t)$$

Memory

RECURRENT NEURAL NETWORKS (RNN)



An unrolled recurrent neural network.

RNN - UNROLLED

$$h_t = \tanh(Wx_t + Uh_{t-1})$$

$$y_t = Vh_t$$

$$E = f_1(y_t), \qquad y_t = f_2(W, U)$$

VANISHING GRADIANT PROBLEM

Back-propagation :

$$W \longleftarrow W - \alpha \frac{\partial E}{\partial W}$$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}}\cdots\frac{\partial c_1}{\partial W} = \quad \frac{\partial E}{\partial y}\frac{\partial y}{\partial c_t}\left[\prod_{i=2}^{i=t}\frac{\partial c_t}{\partial c_{t-1}}\right]\frac{\partial c_1}{\partial W}$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \tanh'(Ux_t + Wc_{t-1})W$$

Derivative of tanh is small so:

$$\left[\prod_{i=2}^{i=t}\frac{\partial c_t}{\partial c_{t-1}}\right] \longrightarrow 0 \quad \textbf{and so} \quad \frac{\partial E}{\partial y} \longrightarrow 0$$

LONG-SHORT-TERM MEMORY (LSTM)

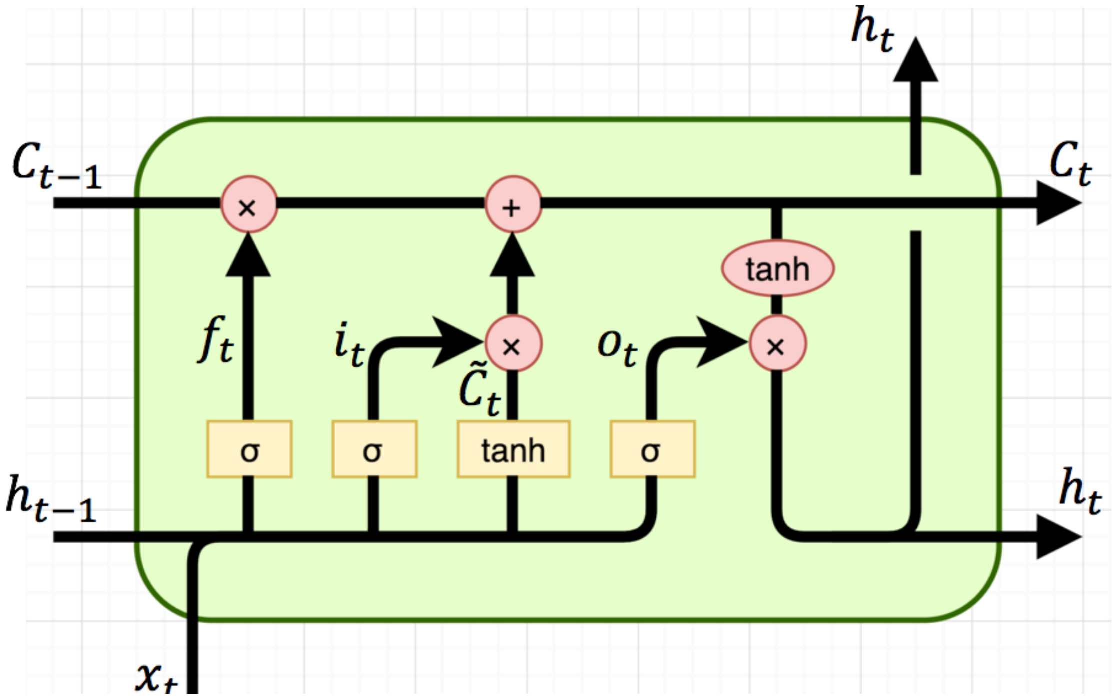
In order to solve vanishing gradient problem Long-Short-Term-Memory (LSTM) were proposed [Hochreiter and Schmidhuber (1997)]

$$i_t = \sigma(U^i x_t + W^i h_{t-1} + b^i)$$

$$f_t = \sigma(U^f x_t + W^f h_{t-1} + b^f)$$

$$\tilde{C}_t = \tanh(U^u x_t + W^u h_{t-1} + b^u)$$

$$o_t = \sigma(U^o x_t + W^o h_{t-1} + b^o)$$



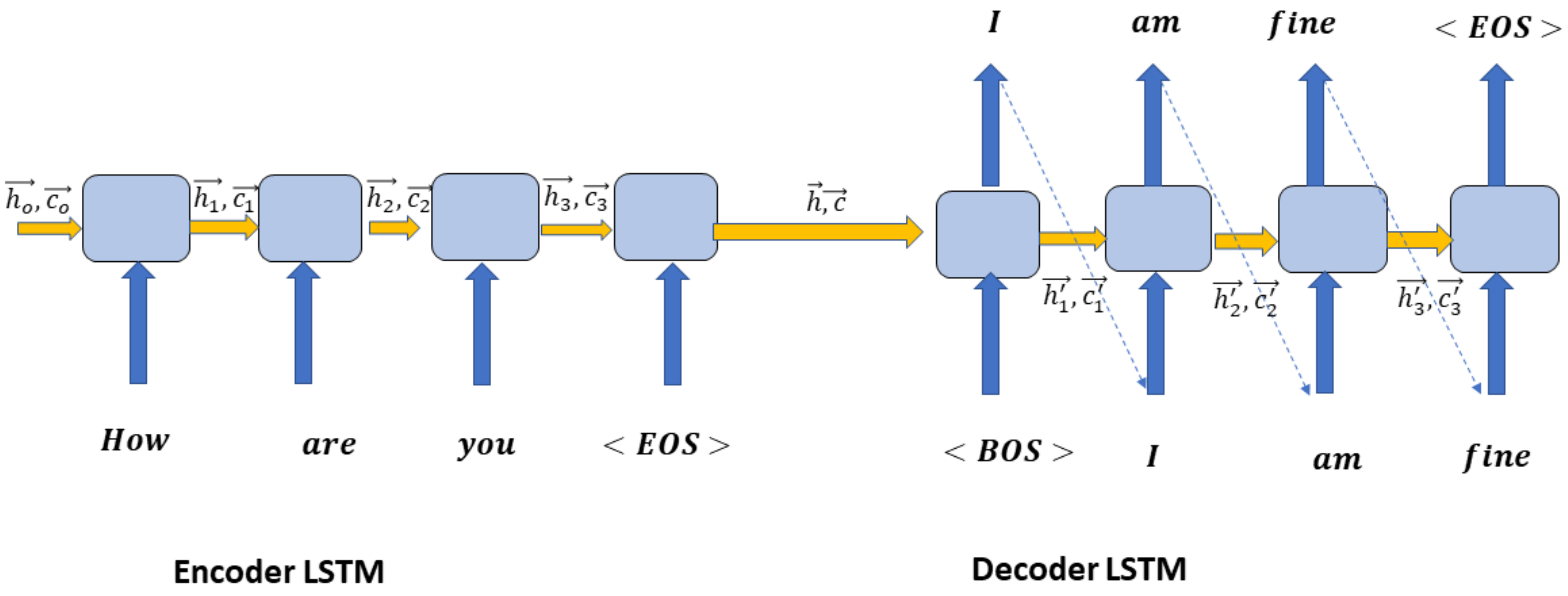
$$C_t = i_t \odot \tilde{C}_t + f_t \odot C_{t-1}$$

$$h_t = o_t \odot \tanh(C_t)$$

$$\frac{\partial C_t}{\partial C_{t-1}} = f_t +$$

An interesting tutorial of LSTM is given here.

ENCODER-DECODER MODEL BASED ON LSTM



$\vec{h}$ → *hidden state of the last input step; context vector*

< EOS > → End of sentence

< BOS > –> Beginning of sentence

Sequence to sequence model

SOURCE CODE MODELLING

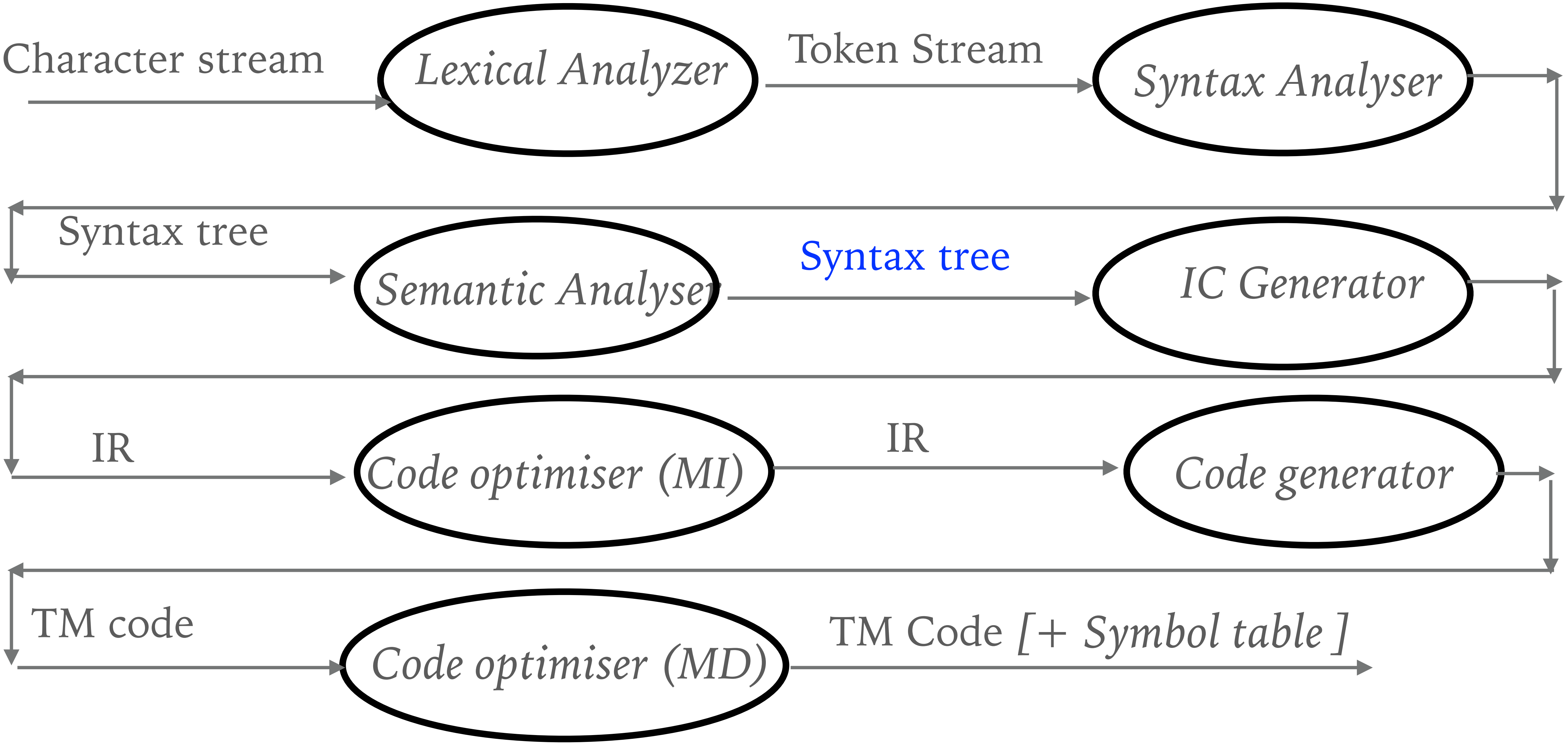
SOURCE CODE

- Source code in its original form is not suitable for feeding to a machine learning model.
- Source code neither follows a linear structure like text (there are loops, conditional jumps, non-local references) nor a fixed vocabulary.
- There are special characters (‘{‘, ‘>’ etc.,) representing the syntax of programming language which create extra difficulties.
- There have been limited success in modelling source code as a linear sequence of tokens.
- One of the common approaches have been to use AST.

ABSTRACT SYNTAX TREE (AST)

- Source code represents the algorithm to convert some input to output.
- Before we get output from a given input using a program the source program must be converted to a ‘target program’ using a compiler [Aho, Lam, Sethi & Ullman (2006)].
- The key tasks of a compiler are program analysis, synthesis and optimisation.
- Once a source code is tokenised it can be converted to a tree like (data) structure called Abstract Syntax Tree (AST), following the grammar of the underlying programming language.

COMPILATION

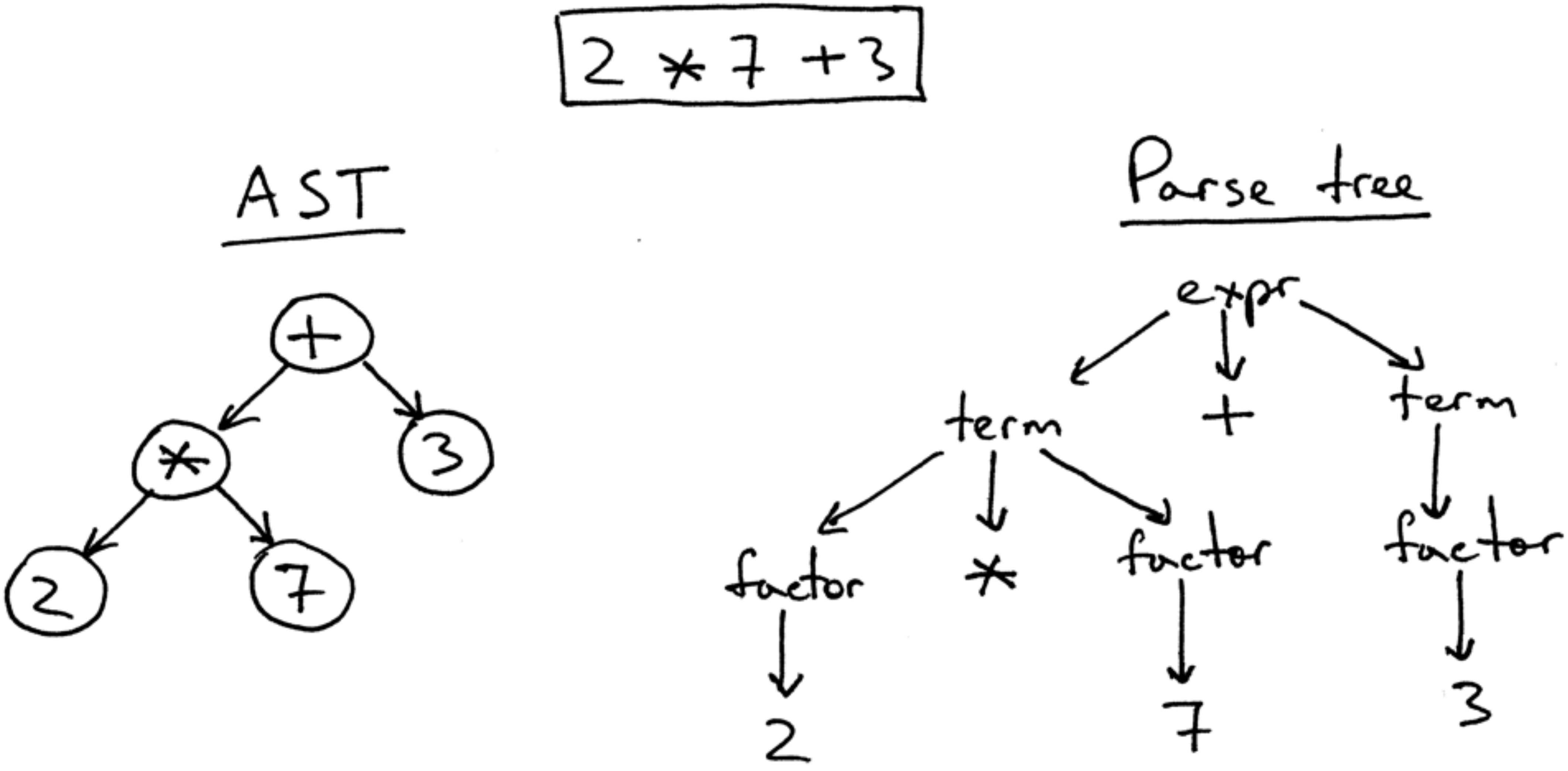


IC-Intermediate Code, MD-Machine Dependent, IR-Intermediate Representation

MI - Machine independent, IC-Intermediate Code, TM - Target Machine

Reference : [Aho, Lam, Sethi & Ullman (2006)]

ABSTRACT SYNATX TREE



Abstract Syntax Tree and Parse Tree

BABELFISH

- **Babelfish** is universal source code parser created by a company named ‘source{d}’
- Babelfish is an open source tool and have drivers available for many popular programming languages such as Python, Java and Go.

“topic extraction from millions of public repositories”

“Programming languages themselves have a limited number of reserved keywords and character based tokens that define the language specification. However, programmers have a rich use of natural language within their code through comments, text literals and naming entities. The programmer defined names that can be found in source code are a rich source of information to build a high level understanding of the project. The goal of this paper is to apply topic modeLling to names used in over 13.6 million repositories and perceive the inferred topics.

[Markovtsev & Kant (2017)]

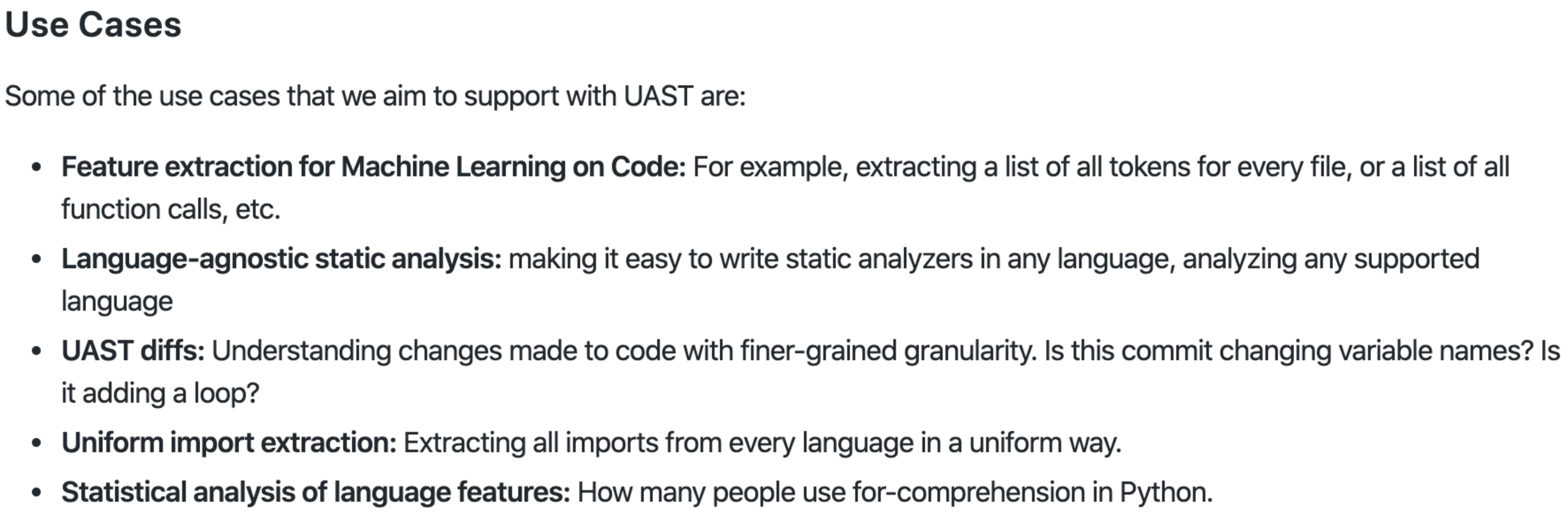
BABELFISH PARSER

- Babelfish is a self hosted server for source code parsing.
- Babelfish can parse any source code file in supported languages and extract an Abstract Syntax Tree (AST) and then an Universal Abstract Syntax Tree (UAST) which is a normalised form of AST.
- Babelfish was developed keeping in mind parsing of a large number of source code files.
- Some of the common fields of a (U)AST node in the tree are- internal type, roles, token and position.

"One of the biggest challenges we've had is how do you understand natural language in code? When we look at the future of search, the future of code suggestion, the future of compilers, it comes a lot down to understanding natural language, understanding what the intent of the developer actually is and what they're trying to do with a piece of code that they're writing,"

- **Eiso Kant, CEO and co-founder at source{d},**

BABEL FISH SERVER



Use Cases

Some of the use cases that we aim to support with UAST are:

- **Feature extraction for Machine Learning on Code:** For example, extracting a list of all tokens for every file, or a list of all function calls, etc.
- **Language-agnostic static analysis:** making it easy to write static analyzers in any language, analyzing any supported language
- **UAST diffs:** Understanding changes made to code with finer-grained granularity. Is this commit changing variable names? Is it adding a loop?
- **Uniform import extraction:** Extracting all imports from every language in a uniform way.
- **Statistical analysis of language features:** How many people use for-comprehension in Python.

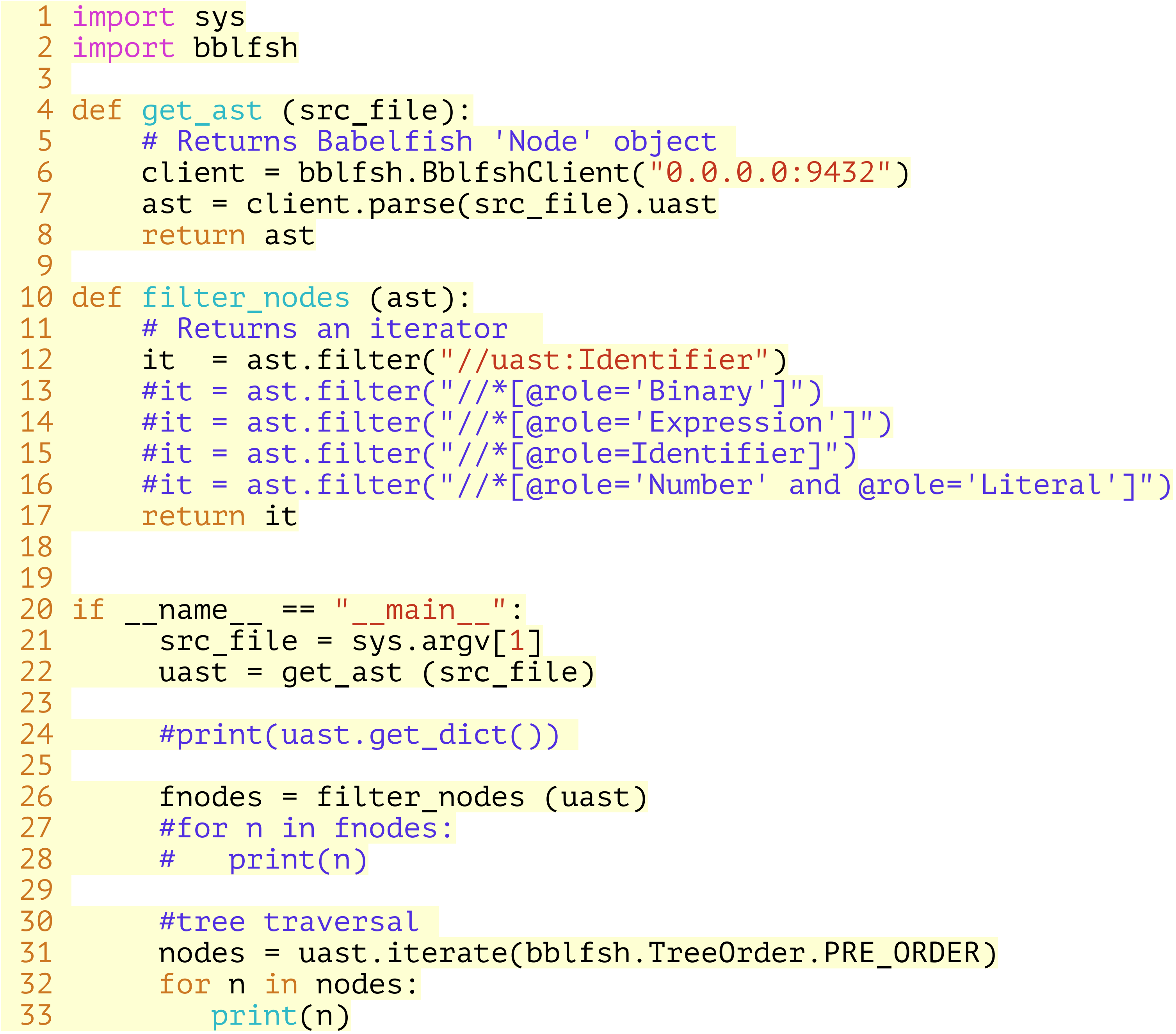
Reference : https://github.com/bblfsh/documentation

Babelfish server : https://github.com/bblfsh/bblfshd

Babelfish python client : https://github.com/bblfsh/python-client

Getting started : https://doc.bblf.sh/using-babelfish/getting-started.html

USING PYTHON CLIENT



```
import sys
import bblfsh

def get_ast (src_file):
    # Returns Babelfish 'Node' object
    client = bblfsh.BblfshClient("0.0.0.0:9432")
    ast = client.parse(src_file).uast
    return ast

def filter_nodes (ast):
    # Returns an iterator
    it  = ast.filter("//uast:Identifier")
    #it = ast.filter("//*[@role='Binary']")
    #it = ast.filter("//*[@role='Expression']")
    #it = ast.filter("//*[@role=Identifier]")
    #it = ast.filter("//*[@role='Number' and @role='Literal']")
    return it


if __name__ == "__main__":
     src_file = sys.argv[1]
     uast = get_ast (src_file)

     #print(uast.get_dict())

     fnodes = filter_nodes (uast)
     #for n in fnodes:
     #   print(n)

     #tree traversal
     nodes = uast.iterate(bblfsh.TreeOrder.PRE_ORDER)
     for n in nodes:
        print(n)
```

UAST NODE OBJECT

Node object has a method name ‘get_dict’ which can be used

To get attributes of a node.

The main attributes are as followings:

1. Internal type : D[’@type’]
2. Position : D[‘@pos’] -> start [line, col], end[start, end]
3. Roles : D[‘@role’]
4. Token : D[‘@token’]
5. Name : D[‘Name’]
6. Children

PROBLEMS

1. Using the UAST given by **Babelfish** find the names of all method call in a program.
2. Find the subtree corresponding to all the methods of a class in A Java file.
3. Get the list of all the internal types and roles used in a program.
4. Write a program to map the UAST nodes (based on start line) to code lines (approximately).
5. Get the list of all UAST nodes which corresponds to binary expressions in a UAST tree using XPATH quarry.

IMPORTANT DECISIONS

- Which of the nodes are relevant for a given machine learning problem ?
- Which properties of UAST nodes are relevant ?
- How to define a ‘data unit’ for machine learning - file tree, class tree, method tree, block tree ?
- Can we use a serialise form of UAST for machine learning - if yes, then how to achieve that ?
- How to consume natural language text/tokens in machine learning ?
- Code2Vec - Embedding ?

LANGUAGE PROCESSING

NATURAL LANGUAGE PROCESSING

- Input to a computing system finally should be in the form of numerical data on which mathematical operations can be applied.
- We can convert a sentence to a set of tokens (words) which can be represented by integer using a lookup table for all the words in the vocabulary of the language being used.
- The above representation is no better than using 'nominal variables' since the numerical value has no meaning - If Mango: 10, Car : 37 does not mean 'Car > Mango'
- One hot vector representation makes more sense.

LANGUAGE PROCESSING

- Let us consider the sentence "This is a car" and assume that we have just four words - "this", "is","a" and "car" in our vocabulary so we can represent these words with the following set of vectors :

  $This = [1,0,0,0], is = [0,1,0,0], a = [0,0,1,0], car = [0,0,0,1]$

- One hot vector representation has certain advantages but still have many shortcomings such as the length of vector becomes very large for rich vocabulary, most of the elements are zeros or the vectors are sparse.

- The best solution is somehow to find "dense vector" representation which uses vectors of smaller dimensions with all the elements have non-trivial values - are non-zeros.

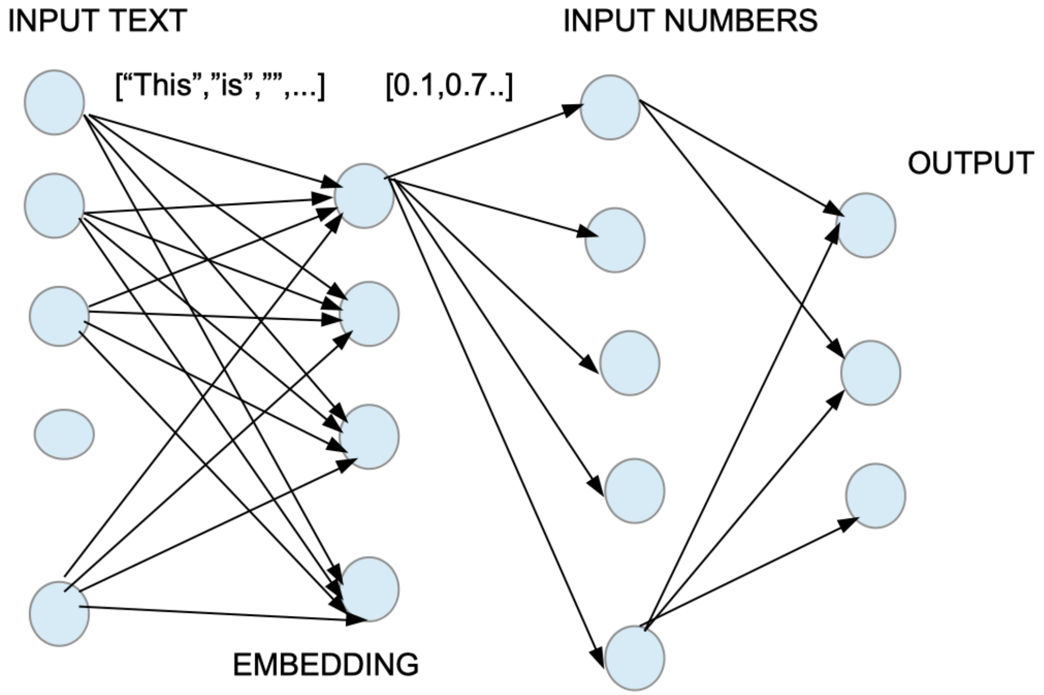
- There are two ways to achieve this : TF-DIF & Embedding.

LANGUAGE PROCESSING

- Term-frequency-inverse-document frequency (tf-idf) is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus [Wikipedia].
- Distributed representation or word embedding : *"a word is characterised by the company it keeps"* - Firth
- Let us consider we have a vocabulary of size ’n’ and we have a set of sentences ’n’ sentences with each of length ‘l’.
- We can represent our data with a matrix ‘X’ of shape $m \times l$ with each element of the matrix between 0 to n.
- Now we can make a linear transformation.

WORD EMBEDDING

- $[Y]_{m\times r} = [X]_{m\times l} \times [A]_{l\times r} \longrightarrow$ Weight matrix
- These scheme depends on computing the matrix A which can be done with neural networks also.



Word Embedding

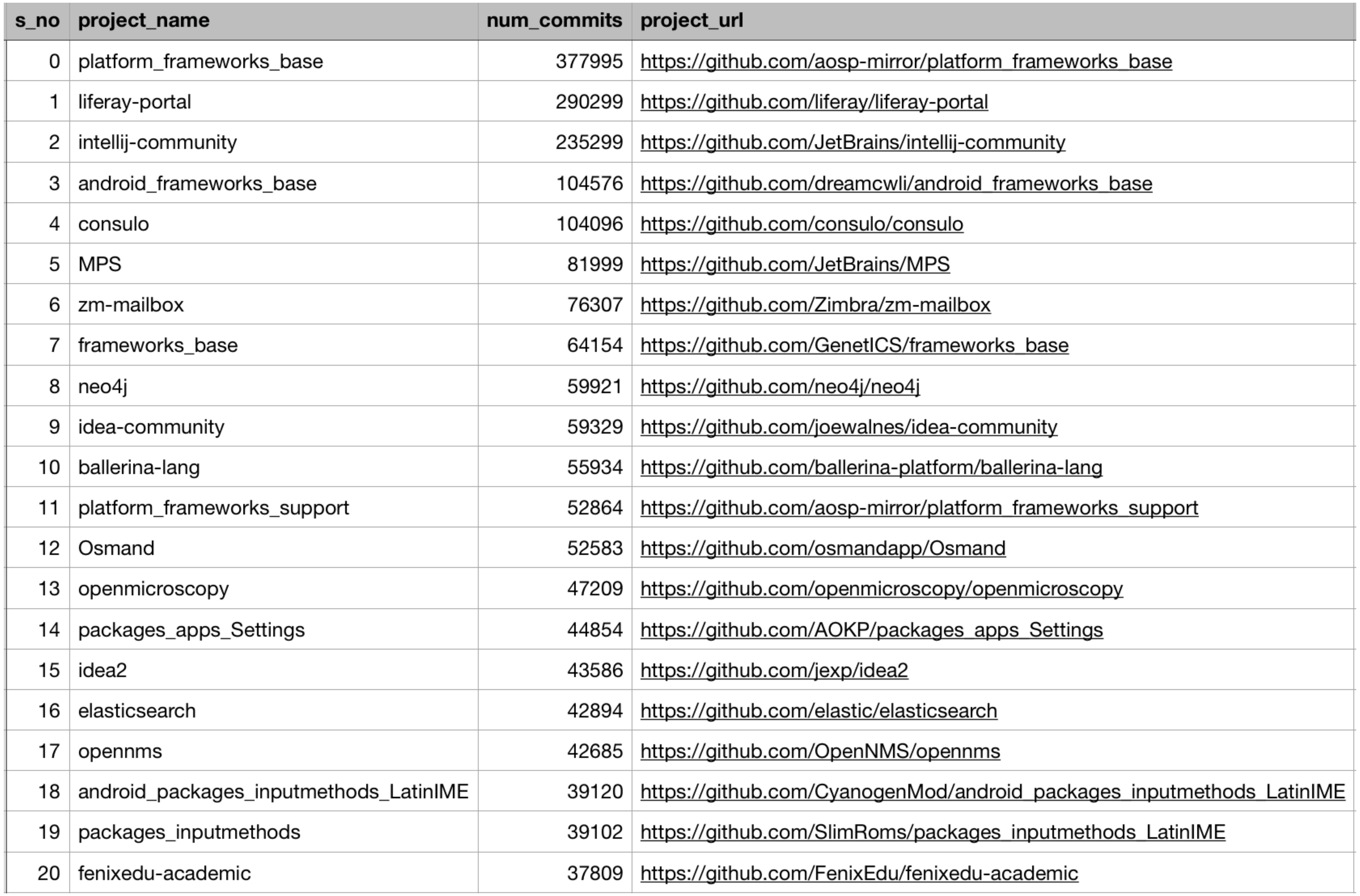
[Glove, Wod2Vec]

DATA CRAWLING

GITHUB

- Github is one of the biggest open source software hosting platform.
- Data from Github can be crawled using the REST API provided by Github.
- We will be using ‘PyGithub==1.45’ for which uses Github API to get the list of repositories for a given language with some attributes (minimum number of commits, stars etc.).
- We will be mostly interested in (open source) repos which are not forked and have source code provided.
- See the program github_data_crawler.py for implementation.

GITHUB (JAVA) REPOS



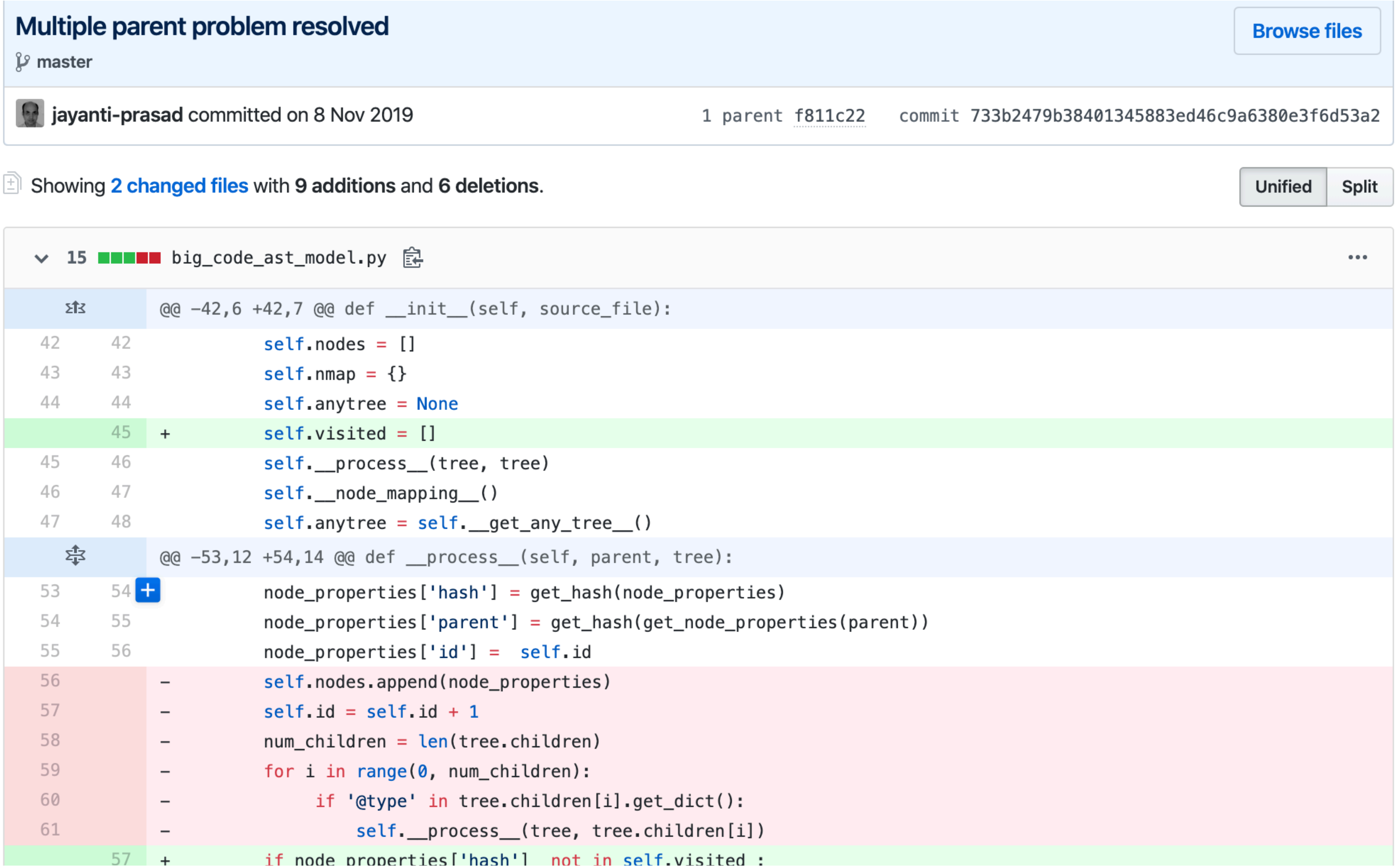
s_no	project_name	num_commits	project_url
0	platform_frameworks_base	377995	https://github.com/aosp-mirror/platform_frameworks_base
1	liferay-portal	290299	https://github.com/liferay/liferay-portal
2	intellij-community	235299	https://github.com/JetBrains/intellij-community
3	android_frameworks_base	104576	https://github.com/dreamcwli/android_frameworks_base
4	consulo	104096	https://github.com/consulo/consulo
5	MPS	81999	https://github.com/JetBrains/MPS
6	zm-mailbox	76307	https://github.com/Zimbra/zm-mailbox
7	frameworks_base	64154	https://github.com/GenetICS/frameworks_base
8	neo4j	59921	https://github.com/neo4j/neo4j
9	idea-community	59329	https://github.com/joewalnes/idea-community
10	ballerina-lang	55934	https://github.com/ballerina-platform/ballerina-lang
11	platform_frameworks_support	52864	https://github.com/aosp-mirror/platform_frameworks_support
12	Osmand	52583	https://github.com/osmandapp/Osmand
13	openmicroscopy	47209	https://github.com/openmicroscopy/openmicroscopy
14	packages_apps_Settings	44854	https://github.com/AOKP/packages_apps_Settings
15	idea2	43586	https://github.com/jexp/idea2
16	elasticsearch	42894	https://github.com/elastic/elasticsearch
17	opennms	42685	https://github.com/OpenNMS/opennms
18	android_packages_inputmethods_LatinIME	39120	https://github.com/CyanogenMod/android_packages_inputmethods_LatinIME
19	packages_inputmethods	39102	https://github.com/SlimRoms/packages_inputmethods_LatinIME
20	fenixedu-academic	37809	https://github.com/FenixEdu/fenixedu-academic

Some of the Big Github Java repos

GIT VERSION CONTROL SYSTEM

- Git is a distributed version control system (vcs) which was developed by Linus Torvalds for linux kernel development.
- Git can be used to ‘clone’ remote repos on a local system and modify those and ‘push’ those to the remote system.
- All the code changes on git repo are in the form of ‘commits’.
- A single git commit may have multiple file changes (add, delete, modified).
- A single file may have multiple code ‘hunks’ changed.
- There is associate information (‘commit message’, ‘author’, ‘date’..) with every commit.

GIT DIFF



Multiple parent problem resolved

master

jayanti-prasad committed on 8 Nov 2019

1 parent f811c22 commit 733b2479b38401345883ed46c9a6380e3f6d53a2

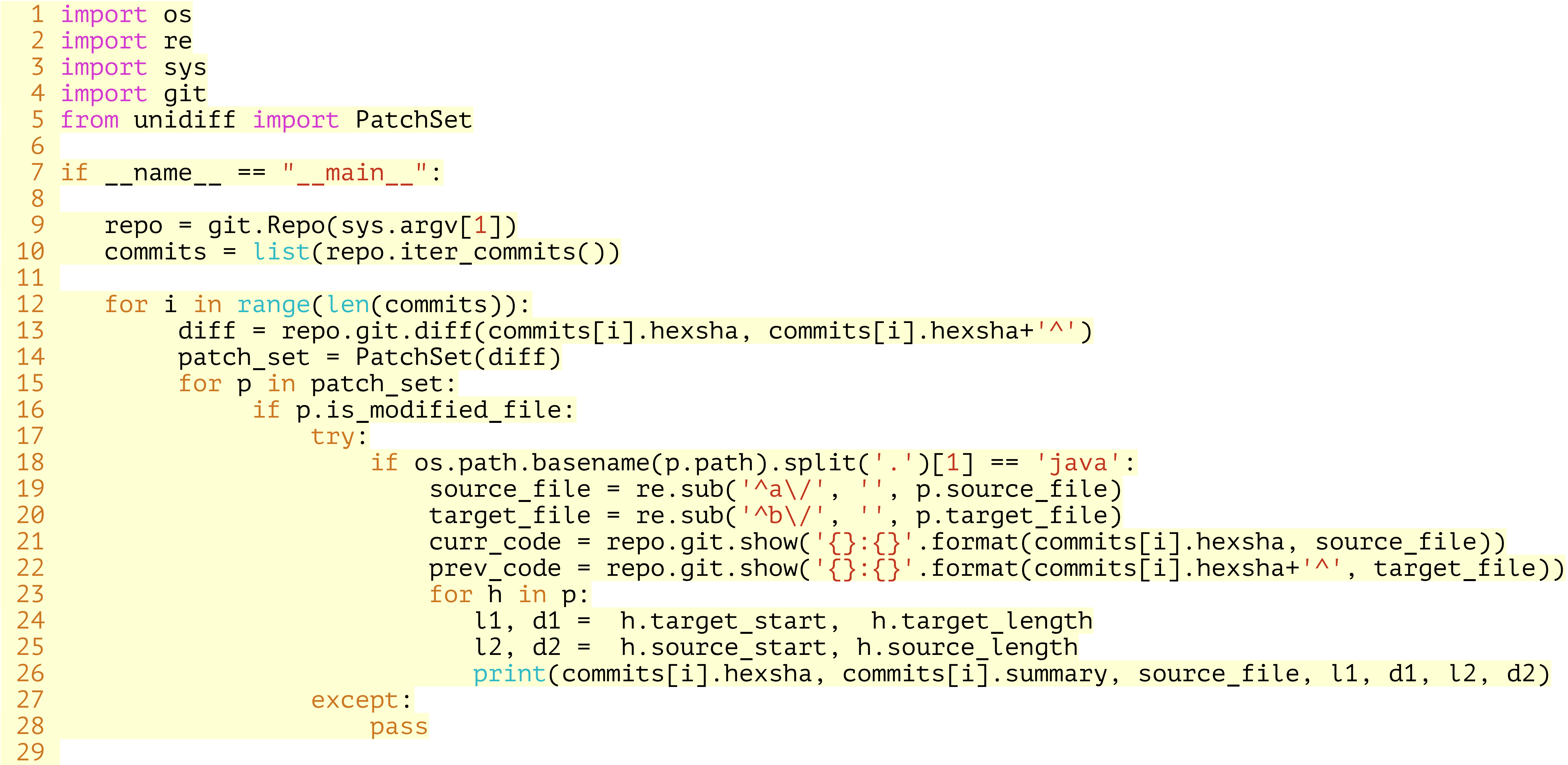
Showing 2 changed files with **9 additions** and **6 deletions**.

15 🟩🟩🟩🟥🟥 big_code_ast_model.py

```
@@ -42,6 +42,7 @@ def __init__(self, source_file):
         self.nodes = []
         self.nmap = {}
         self.anytree = None
+        self.visited = []
         self.__process__(tree, tree)
         self.__node_mapping__()
         self.anytree = self.__get_any_tree__()
@@ -53,12 +54,14 @@ def __process__(self, parent, tree):
         node_properties['hash'] = get_hash(node_properties)
         node_properties['parent'] = get_hash(get_node_properties(parent))
         node_properties['id'] =  self.id
-        self.nodes.append(node_properties)
-        self.id = self.id + 1
-        num_children = len(tree.children)
-        for i in range(0, num_children):
-             if '@type' in tree.children[i].get_dict():
-                 self.__process__(tree, tree.children[i])
+        if node_properties['hash']  not in self.visited :
```

GIT COMMIT PROCESSING

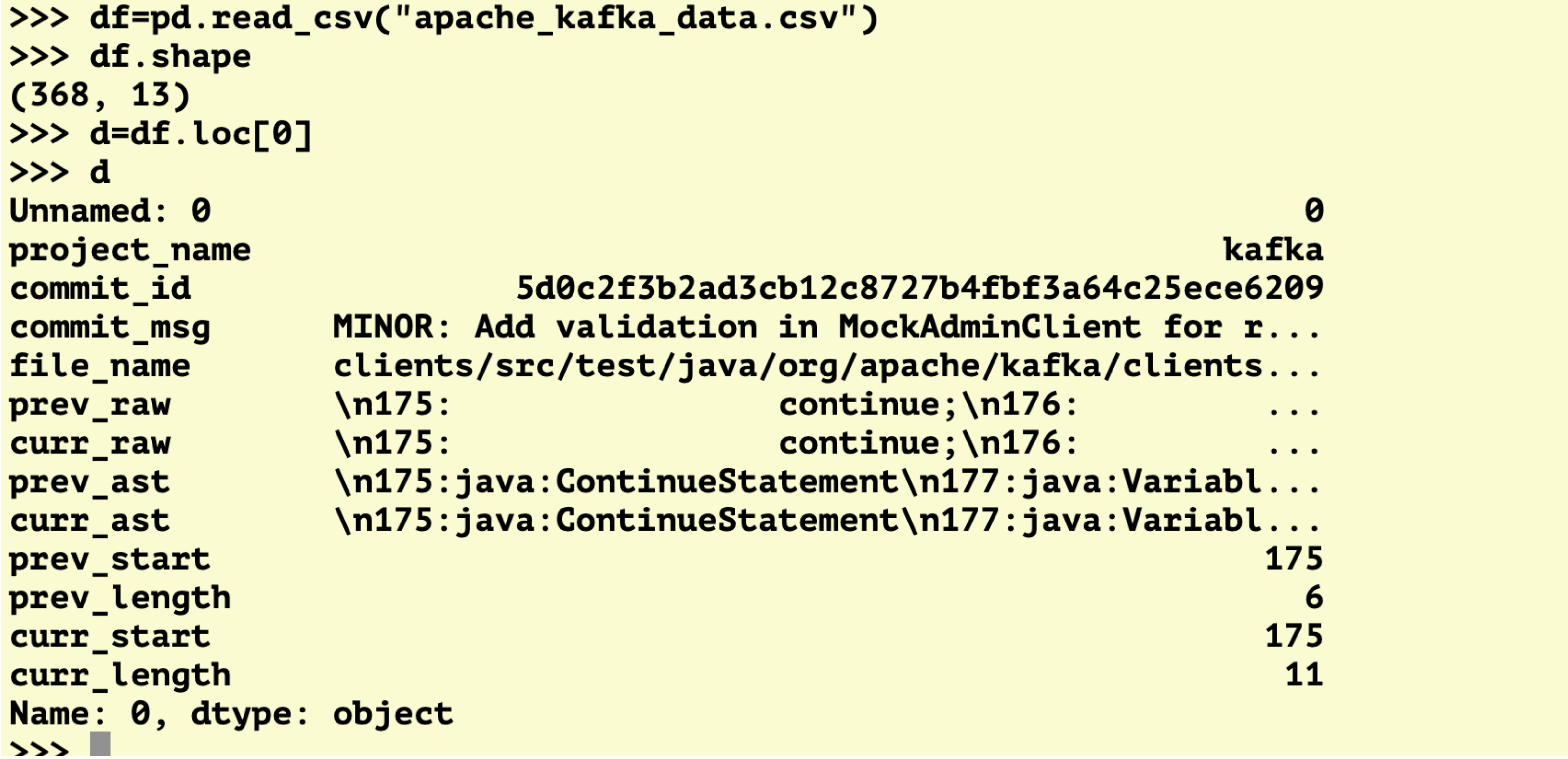
Program : git_tester.py



```python
import os
import re
import sys
import git
from unidiff import PatchSet

if __name__ == "__main__":

   repo = git.Repo(sys.argv[1])
   commits = list(repo.iter_commits())

   for i in range(len(commits)):
        diff = repo.git.diff(commits[i].hexsha, commits[i].hexsha+'^')
        patch_set = PatchSet(diff)
        for p in patch_set:
             if p.is_modified_file:
                 try:
                     if os.path.basename(p.path).split('.')[1] == 'java':
                         source_file = re.sub('^a\/', '', p.source_file)
                         target_file = re.sub('^b\/', '', p.target_file)
                         curr_code = repo.git.show('{}:{}'.format(commits[i].hexsha, source_file))
                         prev_code = repo.git.show('{}:{}'.format(commits[i].hexsha+'^', target_file))
                         for h in p:
                            l1, d1 =  h.target_start,  h.target_length
                            l2, d2 =  h.source_start, h.source_length
                            print(commits[i].hexsha, commits[i].summary, source_file, l1, d1, l2, d2)
                 except:
                     pass

```

GIT DIFF DATA

We can process a git hub repo & can create data in a csv form with the following columns :

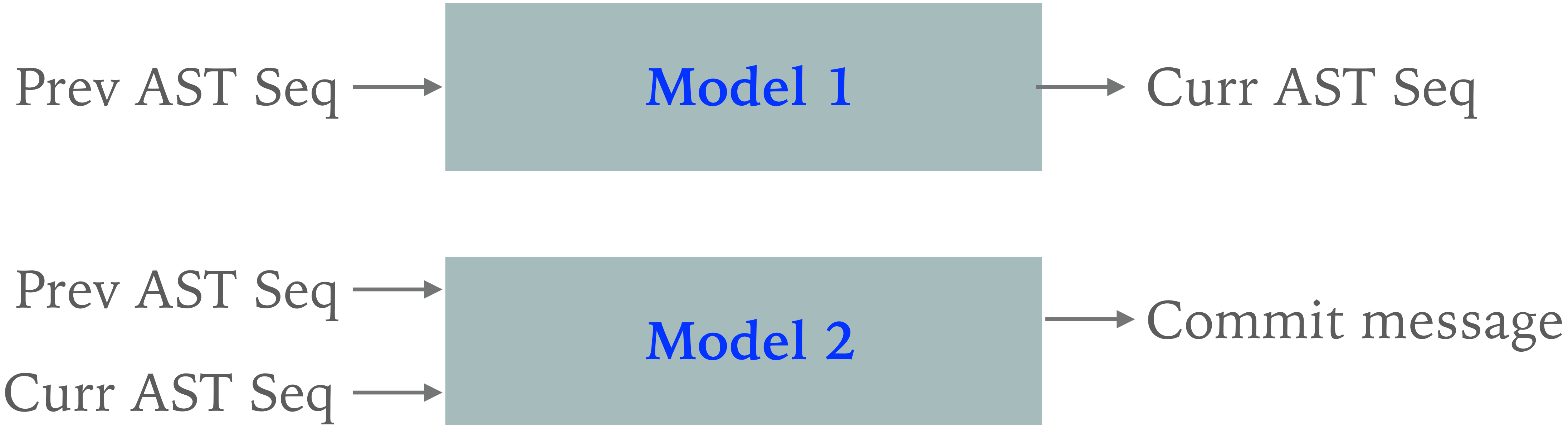


```
>>> df=pd.read_csv("apache_kafka_data.csv")
>>> df.shape
(368, 13)
>>> d=df.loc[0]
>>> d
Unnamed: 0                                                      0
project_name                                                kafka
commit_id                5d0c2f3b2ad3cb12c8727b4fbf3a64c25ece6209
commit_msg      MINOR: Add validation in MockAdminClient for r...
file_name       clients/src/test/java/org/apache/kafka/clients...
prev_raw        \n175:                continue;\n176:         ...
curr_raw        \n175:                continue;\n176:         ...
prev_ast        \n175:java:ContinueStatement\n177:java:Variabl...
curr_ast        \n175:java:ContinueStatement\n177:java:Variabl...
prev_start                                                    175
prev_length                                                     6
curr_start                                                    175
curr_length                                                    11
Name: 0, dtype: object
>>> 
```

A typical data unit (row)

SEQUENCE 2 SEQUENCE MODEL

- For supervised learning with need input data (source) & output data (target).
- Neural network allow multiple inputs & outputs so we can select any number of columns as inputs and any number of columns as output.



[Sutskever et. al. (2014)]

SEQSEQ MODEL-1

Driver Program

```python
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='cmod')
    parser.add_argument('-itn', '--num_input_tokens',type=int,
        help='Number of input tokens', required=True)
    parser.add_argument('-otn', '--num_output_tokens',type=int,
        help='Number of output tokens', required=True)
    parser.add_argument('-isl', '--len_input_seq',type=int,
        help='Length of input sequence', required=True)
    parser.add_argument('-osl', '--len_output_seq',type=int,
        help='Length of input sequence', required=True)
    parser.add_argument('-ldm', '--latent_dim',type=int,
        help='Latent dimension', required=True)
    parser.add_argument('-n', '--epoch', type=int,
        help='Epochs', required=True)

    cfg = parser.parse_args()

    encoder_model, encoder_inputs, encoder_outputs  = get_encoder_model (cfg)

    print(encoder_model.summary())
    utils.plot_model(encoder_model, to_file = "encoder.png")

    model = get_model (cfg, encoder_inputs, encoder_outputs)
    print(model.summary())
    utils.plot_model(model, to_file = "model.png")


```

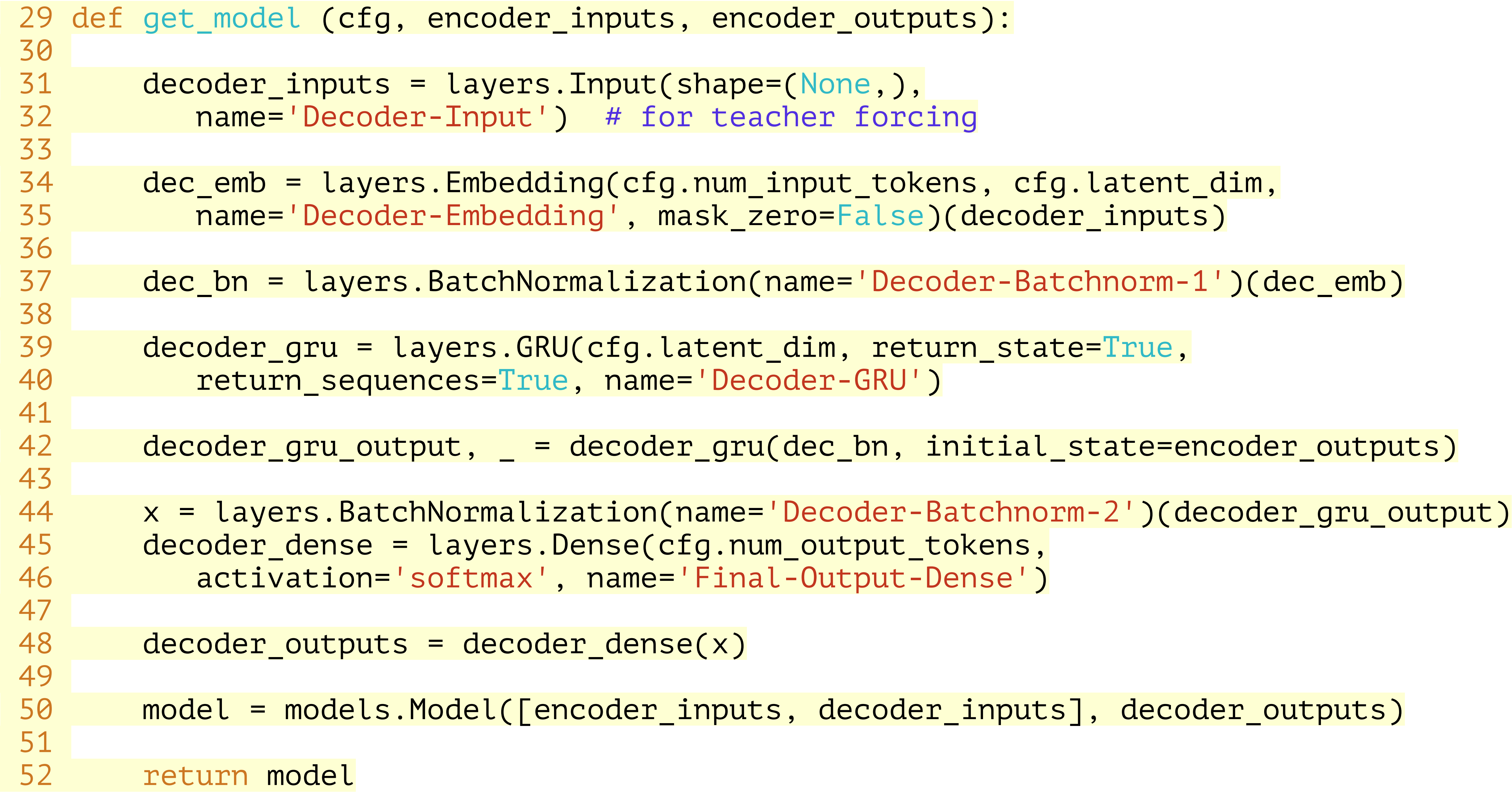
SEQ2SEQ MODEL

Encoder Model

```python
import sys
import argparse
import tensorflow.compat.v1.keras.layers as layers
import tensorflow.compat.v1.keras.models as models
import tensorflow.compat.v1.keras.utils  as utils
import tensorflow.compat.v1.keras.optimizers as optimizers
import tensorflow.compat.v1.keras.callbacks as callbacks

def get_encoder_model (cfg):
    encoder_inputs  = layers.Input(shape=(cfg.len_input_seq,),
       name='Encoder-Input')

    x = layers.Embedding(cfg.num_input_tokens, cfg.latent_dim,
       name='Encoder-Embedding', mask_zero=False) (encoder_inputs)

    x = layers.BatchNormalization(name='Encoder-Batchnorm-1')(x)

    _, state_h = layers.GRU(cfg.latent_dim, return_state=True,\
       name='Encoder-Last-GRU')(x)

    encoder_model = models.Model(inputs=encoder_inputs,
       outputs=state_h, name='Encoder-Model')

    encoder_outputs = encoder_model(encoder_inputs)

    return encoder_model, encoder_inputs, encoder_outputs
```

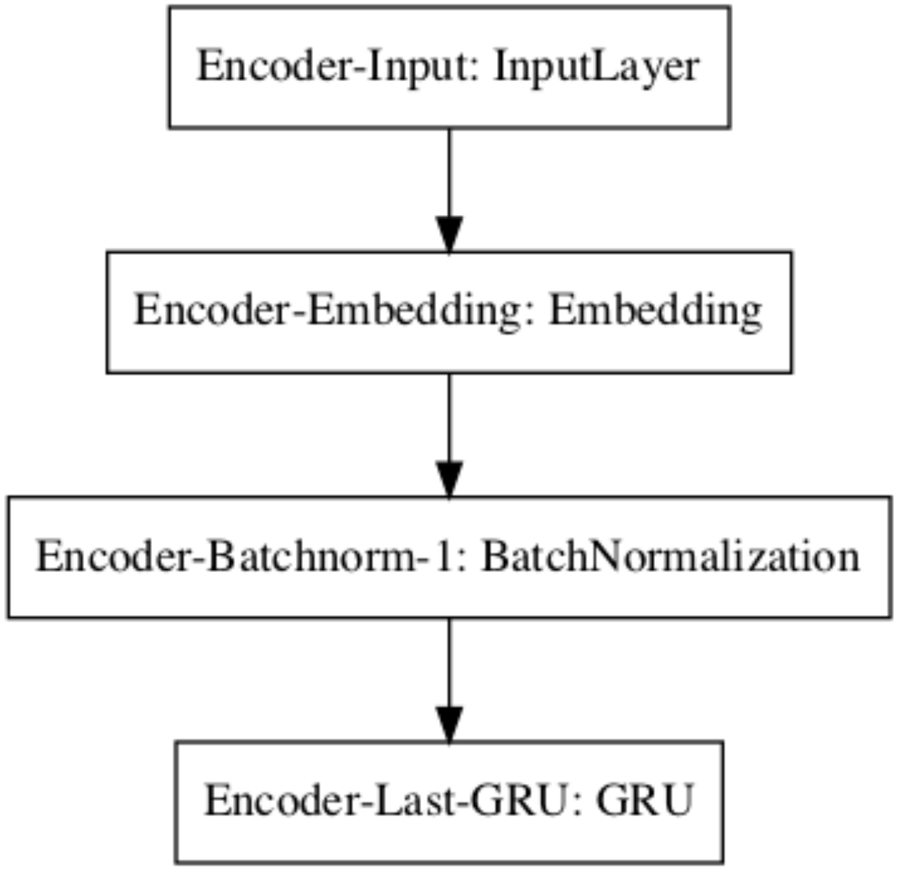
SEQ2SEQ MODEL

Encoder Decoder Model



```python
def get_model (cfg, encoder_inputs, encoder_outputs):

    decoder_inputs = layers.Input(shape=(None,),
       name='Decoder-Input')  # for teacher forcing

    dec_emb = layers.Embedding(cfg.num_input_tokens, cfg.latent_dim,
       name='Decoder-Embedding', mask_zero=False)(decoder_inputs)

    dec_bn = layers.BatchNormalization(name='Decoder-Batchnorm-1')(dec_emb)

    decoder_gru = layers.GRU(cfg.latent_dim, return_state=True,
       return_sequences=True, name='Decoder-GRU')

    decoder_gru_output, _ = decoder_gru(dec_bn, initial_state=encoder_outputs)

    x = layers.BatchNormalization(name='Decoder-Batchnorm-2')(decoder_gru_output)
    decoder_dense = layers.Dense(cfg.num_output_tokens,
       activation='softmax', name='Final-Output-Dense')

    decoder_outputs = decoder_dense(x)

    model = models.Model([encoder_inputs, decoder_inputs], decoder_outputs)

    return model
```

MODEL ARCHITECTURE



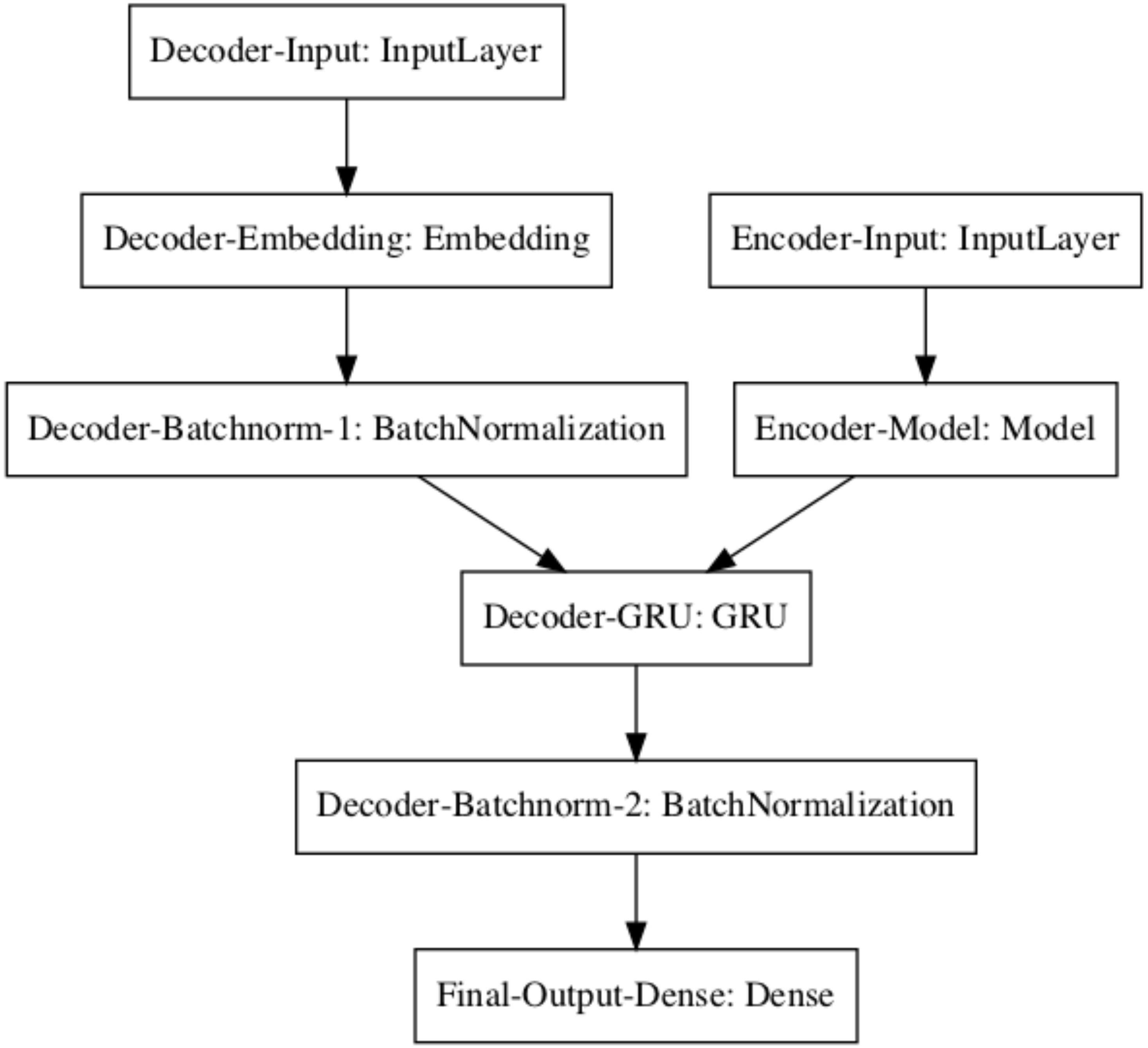
Encoder-Input: InputLayer

Encoder-Embedding: Embedding

Encoder-Batchnorm-1: BatchNormalization

Encoder-Last-GRU: GRU

Encoder Model



Decoder-Input: InputLayer

Decoder-Embedding: Embedding

Encoder-Input: InputLayer

Decoder-Batchnorm-1: BatchNormalization

Encoder-Model: Model

Decoder-GRU: GRU

Decoder-Batchnorm-2: BatchNormalization

Final-Output-Dense: Dense

Encoder-Decoder Model

Sequence to vector utility

Sequence to sequence utility

FITTING THE MODEL

```
def fit_model (cfg, model, X, Y):

    model.compile(optimizer=optimizers.Nadam(lr=0.01),
           loss='sparse_categorical_crossentropy',metrics=['acc'])

    encoder_input_data = X
    decoder_input_data = Y[:, :-1]
    decoder_output_data = Y[:, 1:]

    history =   model.fit([encoder_input_data,
                decoder_input_data],  np.expand_dims(decoder_output_data, -1),
                batch_size =100,
                epochs = cfg.epoch, validation_split = 0.12)

    return history
```

```
    # create fake data
    X  =  np.random.randint(cfg.num_input_tokens,
       size=(1000, cfg.len_input_seq))
    Y  =  np.random.randint(cfg.num_output_tokens,
       size=(1000, cfg.len_output_seq))

    print(X.shape)
    print(Y.shape)

    # fit the model
    h = fit_model (cfg, model, X, Y)
```

INFERENCE

- Read the input trained model & encoder model.
- Build the decoder model.
- Get the encoder ‘state’ for a given input sequence.
- With decoder output as ‘start’ token and encoder input state predict the 1st token and ‘state’.
- Predict the next ‘token’ using the 1st token & state and update state.
- Keep iterating till ‘stop’ token is predicted or maximum sequence length is reached.

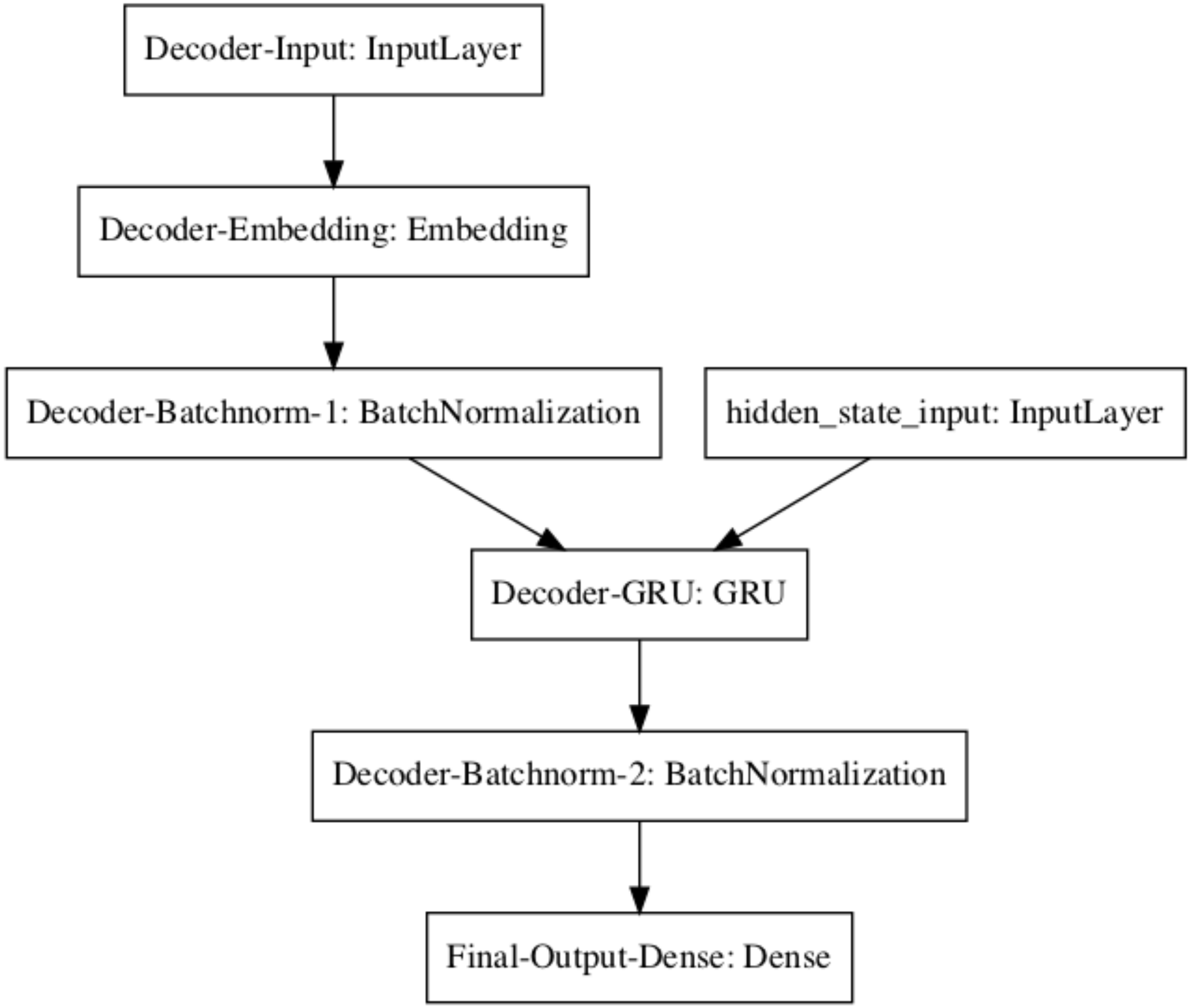
INFERENCE

```python
def get_decoder_model (model):

    latent_dim = model.get_layer('Decoder-Embedding').output_shape[-1]

    decoder_inputs = model.get_layer('Decoder-Input').input
    dec_emb = model.get_layer('Decoder-Embedding')(decoder_inputs)
    dec_bn = model.get_layer('Decoder-Batchnorm-1')(dec_emb)

    gru_inference_state_input = layers.Input(shape=(latent_dim,),
       name='hidden_state_input')

    gru_out, gru_state_out = model.get_layer('Decoder-GRU')
      ([dec_bn, gru_inference_state_input])

    dec_bn2 = model.get_layer('Decoder-Batchnorm-2')(gru_out)
    dense_out = model.get_layer('Final-Output-Dense')(dec_bn2)
    decoder_model = models.Model([decoder_inputs, gru_inference_state_input],
                                   [dense_out, gru_state_out])
    return decoder_model


def load_model (cfg):

    model = models.load_model(cfg.model_file)

    encoder_model = model.get_layer('Encoder-Model')

    decoder_model = get_decoder_model (model)

    return encoder_model, decoder_model, model

```

INFERENCE

```
def predict_seq (cfg, encoder_model, decoder_mode, X):

    start_token = 0
    start_token = 10

    embd_vec = encoder_model.predict(X)

    state_value =  start_token

       decoded_sentence = []
       stop_condition = False

       while not stop_condition:

            preds, st = decoder_model.predict([state_value, embd_vec])

            # We are going to ignore indices 0 (padding) and indices 1 (unknown)
            # Argmax will return the integer index corresponding to the
            #  prediction + 2 b/c we chopped off first two

            pred_idx = np.argmax(preds[:, :, 2:]) + 2

            if pred_idx== end_token  or len(decoded_sentence) >=  cfg.max_target_seq:
                 stop_condition = True
                 break
            decoded_sentence.append(pred_idx)

            # update the decoder for the next word
            embd_vec = st
            state_value = np.array(pred_idx).reshape(1, 1)

        return decoded_sentence
```

DECODER ARCHITECTURE


Decoder-Input: InputLayer
Decoder-Embedding: Embedding
Decoder-Batchnorm-1: BatchNormalization
hidden_state_input: InputLayer
Decoder-GRU: GRU
Decoder-Batchnorm-2: BatchNormalization
Final-Output-Dense: Dense

Decoder Architecture

SUMMARY

- Source code does have some similarities with the text in natural languages so language modelling can be applied on source code repos at massive scale [Allamanis (2013)].
- Github which hosts millions of open source projects (billion of lines of code) can be the ultimate source for data mining & applying machine learning on source code.
- Machine learning on source code can be used to identify useful patterns in source code that can be used in software development in different ways such as identifying risky commits, bug & defect prediction, security vulnerabilities, code summarisation, text generation etc., [Allamanis (2018)].

SUMMARY

➤ Neural machine translation models based on LSTM have been used on the AST of source code to find useful patterns:

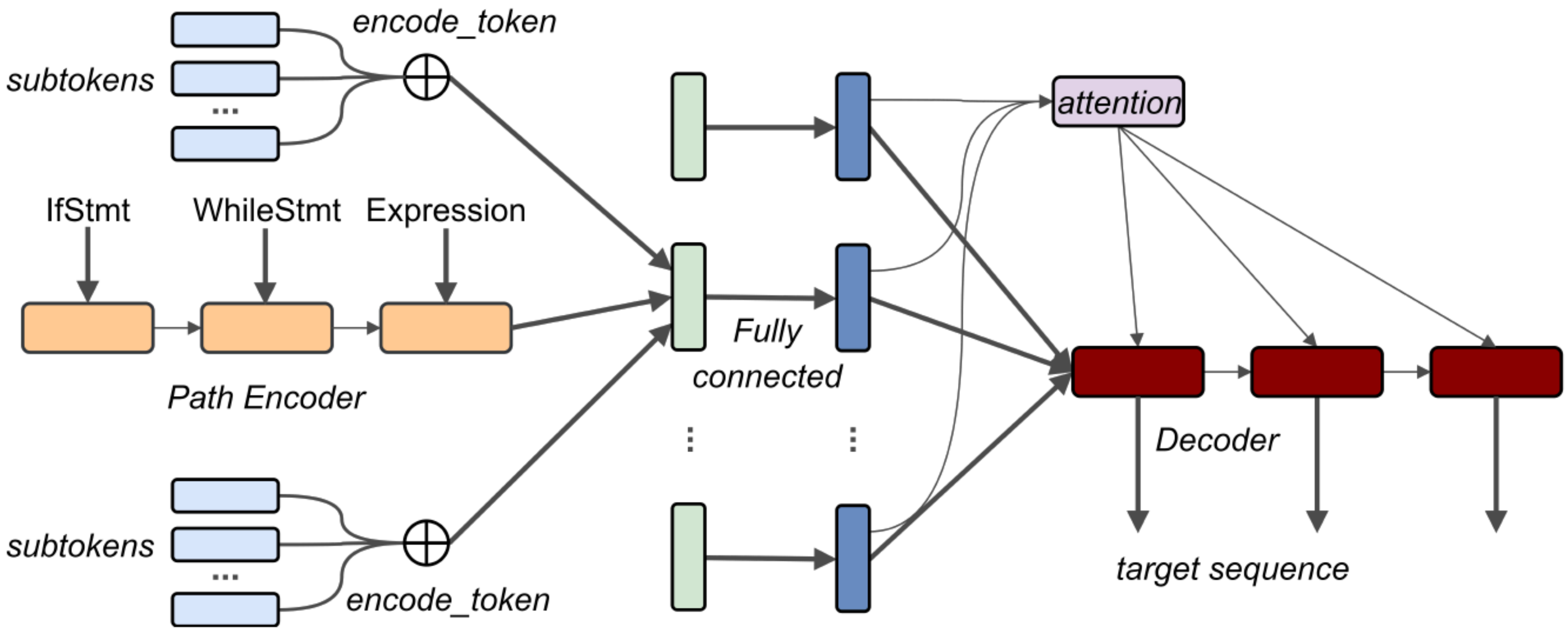
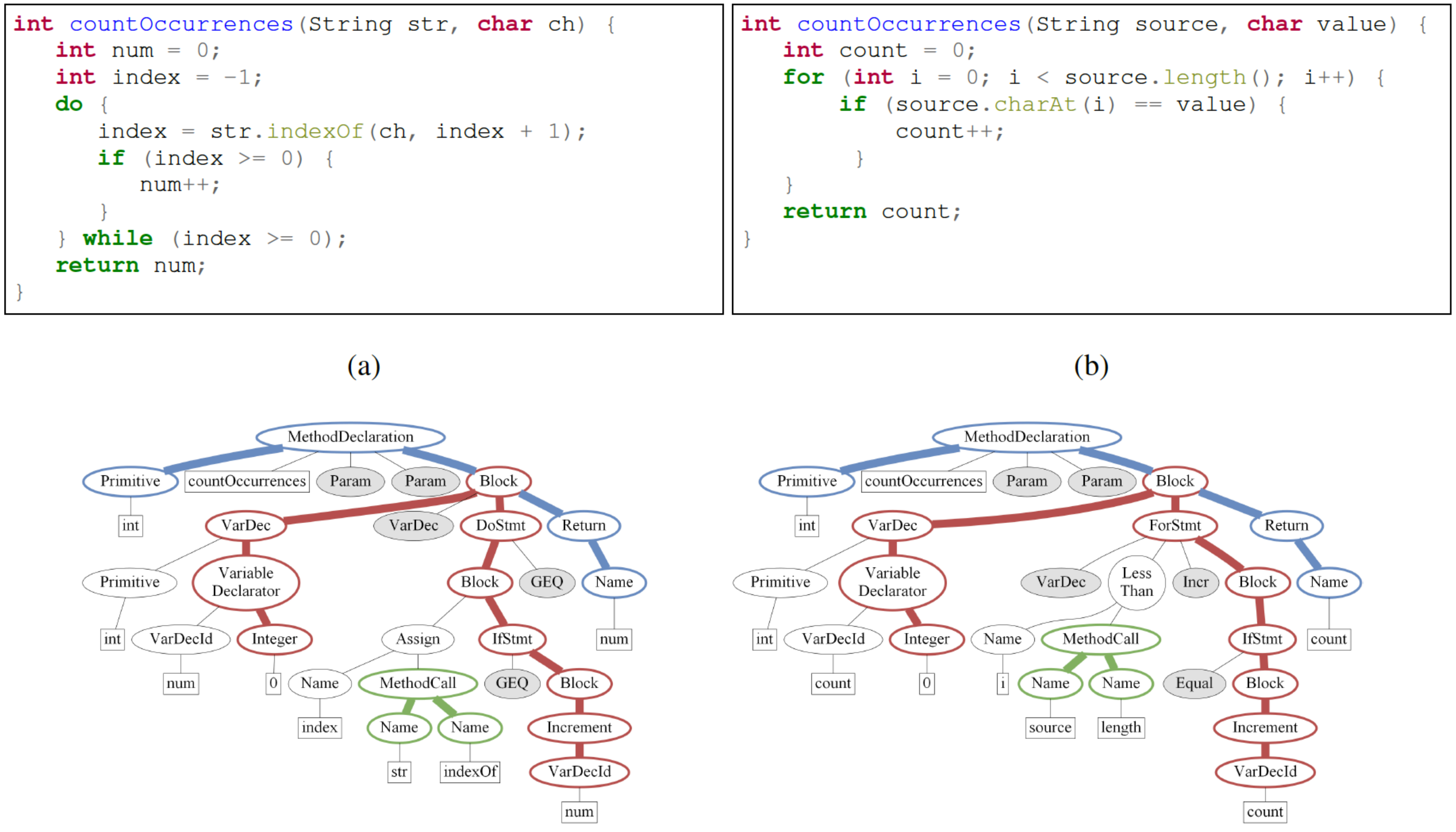


Figure 3: Our model encodes each path as a sequence of AST nodes, and averages the produced input vectors as the initial state of the decoder. The decoder generates an output sequence while attending over the encoded paths.

[Alon et. al. (2018a, 2018b)]

SUMMARY



```
int countOccurrences(String str, char ch) {
    int num = 0;
    int index = -1;
    do {
        index = str.indexOf(ch, index + 1);
        if (index >= 0) {
            num++;
        }
    } while (index >= 0);
    return num;
}
```

(a)

```
int countOccurrences(String source, char value) {
    int count = 0;
    for (int i = 0; i < source.length(); i++) {
        if (source.charAt(i) == value) {
            count++;
        }
    }
    return count;
}
```

(b)

(c)

(d)

Figure 2: An example of two Java methods that have exactly the same functionality. Although having a different *sequential* (token-based) representation, considering syntactic patterns reveals recurring paths, which might differ only in single nodes (a `ForStmt` node instead of a `Do`-while node).

[Alon & Levy (2018)]

SUMMARY

- Some interesting tools have been developed to apply machine learning on source code [Markovtsev & Kant (2017)].
- There have been proposed different approaches to model source code such as sequence tokens, trees & graphs [Brockschmidt et. al. (2018)].
- There have been studies to model source code change based on Tree2Tree models inspired from Seq2Seq model [Chakraborty et. al. (2018a, 2018b)].
- Bug fixing patches have also been generated using NMT models on source code [Michele (2018)].

CONCLUSIONS

- Machine learning on source code is a very promising area of research, however, we still have to see breakthroughs as we have seen in Natural Language Processing.
- This course has just introduced the field & a particular approach to the problem.
- Use of data from the open source repositories and machine learning we may see major developments in the software development process as we have seen in other industries.

REFERENCES

- Aho, Lam, Sethi & Ullman (2006), Compilers, principles, techniques & tools (Addison-Wesely).
- Markovtsev & Kant (2017), Topic modelling of public repositories at scales using names in source code, [arXiv:1704.00135v2].
- Sutskever et. al. (2014), Sequence to Sequence Learning with neural Networks, [arXiv: 14093215].
- Alon et. al. (2018a), code2seq: Generating Sequences from Structured Representations of Code, [arXiv: 1808.01400].
- Alon et. al. (2018b), code2vec, Learning distributed representation of code, ar[Xiv: 1803.09473].

REFERENCES

- Brockschmidt et. al. (2018), Code modelling with graph, [arXiv: 1805.08490].
- Chakraborty et. al. (2018a), Tree2Tree Neural Translation Model for Learning Source Code Changes, [arXiv:1810.00314].
- Chakraborty et. al. (2018b), CODIT: Code Editing with Tree-Based NeuralMachine Translation, arXiv: [arXiv:1810.00314]

THANK YOU !