

**Faculty of Natural and Mathematical Sciences**

# Department of Informatics

King's College London, Strand Campus, London, United Kingdom

7CCSMPRJ

Individual Project Submission 2021/22

Name: Olubusayo Akeredolu

Student Number: 20107125

Degree Programme: MSc Artificial Intelligence

Project Title: Text vs Trees vs Graphs: Deep Learning Techniques for Program Understanding

Supervisor: Dr. Maria Polukarov

Word count: Must be less than 15,000

**RELEASE OF PROJECT**

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

I **agree** to the release of my project

I **do not** agree to the release of my project

**Signature:**



Date:



Department of Informatics

King’s College London

United Kingdom

7CCSMPRJ MSc Project

TEXT VS TRESS VS GRAPHS:

DEEP LEARNING TECHNIQUES FOR PROGRAM UNDERSTANDING

Name: Olubusayo Akeredolu

Student Number: 20107125

Degree Programme: MSc Artificial Intelligence

Supervisor’s Name: Dr. Maria Polukarov

This dissertation is submitted for the degree of MSc in Artificial Intelligence

Acknowledgement

It is a short paragraph to thank those whose have contributed to the project work.

Abstract

It is a precis of the report (normally in one page), which should include:

* A brief introduction to the project objectives
* A brief description of the main work of the project
* A brief description of the contributions, major findings, results achieved and principal conclusion of the project

Nomenclature

*a* The number of angels per unit area

*A* The area of the needle point

*c* Speed of light in a vacuum inertial frame

*h* Planck constant

LMI Linear Matrix Inequalities

*N* The number of angels per needle point

Contents

[1 Introduction 7](#_Toc106362520)

[1.1 Project Overview 7](#_Toc106362521)

[1.2 Aims and Objectives 8](#_Toc106362522)

[1.3 Background and Literature Review 10](#_Toc106362523)

[1.4 Motivations 11](#_Toc106362524)

[2 Background Theories 12](#_Toc106362525)

[3 Objectives, Specifications and Design 13](#_Toc106362526)

[3.1 Specific Project Objectives 13](#_Toc106362527)

[3.1.1 Classification Categories 13](#_Toc106362528)

[3.2 Technical Project Specifications 14](#_Toc106362529)

[3.2.1 Models 14](#_Toc106362530)

[3.2.2 Datasets 14](#_Toc106362531)

[3.3 Design 15](#_Toc106362532)

[3.3.1 Text-Based LSTM Model 15](#_Toc106362533)

[3.3.2 Text-Based FFN Model 16](#_Toc106362534)

[3.3.3 Graph-Based FFN Model 16](#_Toc106362535)

[3.3.4 Graph-Based LSTM Model 16](#_Toc106362536)

[3.3.5 Tree-Based FFN Model 16](#_Toc106362537)

[3.3.6 Tree-Based LSTM Model 16](#_Toc106362538)

[4 Methodology and Implementation 17](#_Toc106362539)

[5 Results, Analysis and Evaluation 18](#_Toc106362540)

[6 Legal, Social, Ethical and Professional Issues 19](#_Toc106362541)

[7 Conclusion 20](#_Toc106362542)

[8 References 21](#_Toc106362543)

[9 Appendices 22](#_Toc106362544)

[9.1 Appendix A: 22](#_Toc106362545)

[9.2 Appendix B: 22](#_Toc106362546)

[9.3 Appendix B: Points to Note 22](#_Toc106362547)

List of Tables

List of Figures

1. Fig 1a
2. Fig 1b
3. Fig 2a
4. Fig 2b

# Introduction

## Project Overview

This project falls under the field of Programming Language Understanding. This is a field of Artificial Intelligence that deals with making use of Machine Learning and Deep Learning techniques to train computers (or intelligent agents) to understand programs, which will be referred to as source code for the purpose of this project.

Programming Language Understanding (PLU) is a highly relevant and interesting field because it examines a very specific way in which neural networks are yet to be as intelligent as human beings: they are unable to differentiate between natural language text and source code. Consider a situation where a Natural Language Processing (NLP) model is trained using source code. The NLP model is designed to work with text and as a result, it will treat the source code in the same way it would treat natural language text. This is a shortcoming because there are many differences between natural language (NL) text and source code.

One of these differences is that natural language generally requires less context in comparison to source code. For example, the natural language English sentence ‘The girl walked away’ is a complete sentence that follows grammatical rules and does not necessarily require more context to be understood. In source code, this is not the case. Each individual line in a program should not and cannot be considered on its own due to the nature of most non-scripting [\_\_] programming languages. Consider, for example, the Java variable declaration ‘int num = a + b’. This statement requires the programmer to consider where the variables ‘a’ and ‘b’ were declared, what their values are, if and how their values change in the program, the data types of these variables, if they are compatible with the ‘+’ operation, the scope or accessibility level of these variables (global or local), etc.

Another important distinction between natural language and source code is the existence of various concepts and structural information in source code, which are absent in natural language. Examples of these concepts include control flow and conditional statements, inheritance and objects in object-oriented programming, abstract classes, and abstraction, etc. Examples of structural information that are present in source code and not in natural language include programming paradigm (imperative, event-driven or declarative languages), typing structure (strongly typed or weakly typed languages), the use of features from imported classes and external libraries, etc.

It is necessary to note that there are different types of programming languages. These include non-scripting languages – procedural languages, object-oriented languages, functional languages – and scripting languages – server-side scripting languages, client-side scripting languages, query languages, etc. It is also important to note that certain languages, e.g., Python, are considered both as scripting and non-scripting languages.

The main difference between scripting and non-scripting languages is that scripting languages are generally interpreted [\_\_], while non-scripting languages are generally compiled [\_\_]. Another important difference to note is that scripting languages tend to be closer in syntax to natural language than non-scripting languages. For example, the SQL (query language) statement ‘SELECT name FROM myTable;’ follows the rules of an English language sentence (it contains an object, a verb, and a subject) and can be understood entirely on its own.

For this reason, this project focuses solely on non-scripting languages, specifically, procedural, and object-oriented languages; Java (object-oriented), Python (procedural and object-oriented) and C (procedural and object-oriented).

## Aims and Objectives

The main objective of this project is to compare deep learning models based on three different information storage structures to determine which of these forms is the most suitable for designing neural networks to carry out classification tasks on source code.

The first of these information storage structures is text. There are many existing deep learning techniques for carrying out learning and classification tasks on text. In order to get the most accurate picture of how generic text-based models work with source code, this project will compare the performances of two different standard NLP classification models. I have chosen these models because they are conventionally used when carrying out learning tasks on text. The models I have chosen are the Long Short-Term Memory (LSTM) neural network and the Feed Forward neural network (FFN).

The second information storage structure is the tree data structure. I have chosen this abstract data type because every program has an Abstract Syntax Tree (AST) [\_\_] representation which shows the structure of the program and the connections present within it. This project involves the development of two neural network models that accept a series of ASTs as their input and carry out learning tasks based on these trees. These neural network architectures will be known as the Tree-Based Feed Forward Neural Network (TBFFN) and the Tree-Based LSTM Neural Network.

The final information storage structure is the graph data structure. I have chosen graphs because every program can be represented as a flowchart or flow graph, which are forms of directed graphs. This project aims to develop the Graph-Based Feed-Forward Neural Network (GFFN) and the Graph-Based LSTM Neural Network which will accept a series of graphs derived from source code as their input and carry out classification tasks on these graphs.

I have chosen to develop models based on trees and graphs because of the differences between source code and text outlined in section 1.1, but also because of the shortcomings of text-based models when processing source code, and the advantages of trees and graphs for capturing structural information. The major shortcoming of text-based models is that when carrying out learning tasks, they convert each individual word, or a series of words (known as n-grams [\_\_]) into a vector. This is an issue when training a network on programs because multiple programs that solve the same problem in the same way might use completely different names to label variables, functions, etc. For example, in fig 1a and 1b, both functions implement the bubble sort algorithm in the exact same way but have their variables and functions named differently.

def bubble\_sort(vector):

for i in range(len(vector)):

for j in range(len(vector) - 1, i, -1):

if vector[j - 1] > vector[j]:

vector[j - 1], vector[j] = vector[j], vector[j - 1]

*fig 1a: An implementation of the Bubble Sort algorithm*

def sort(array):

for a in range(0, len(array)):

for b in range(len(array)-1, a, -1):

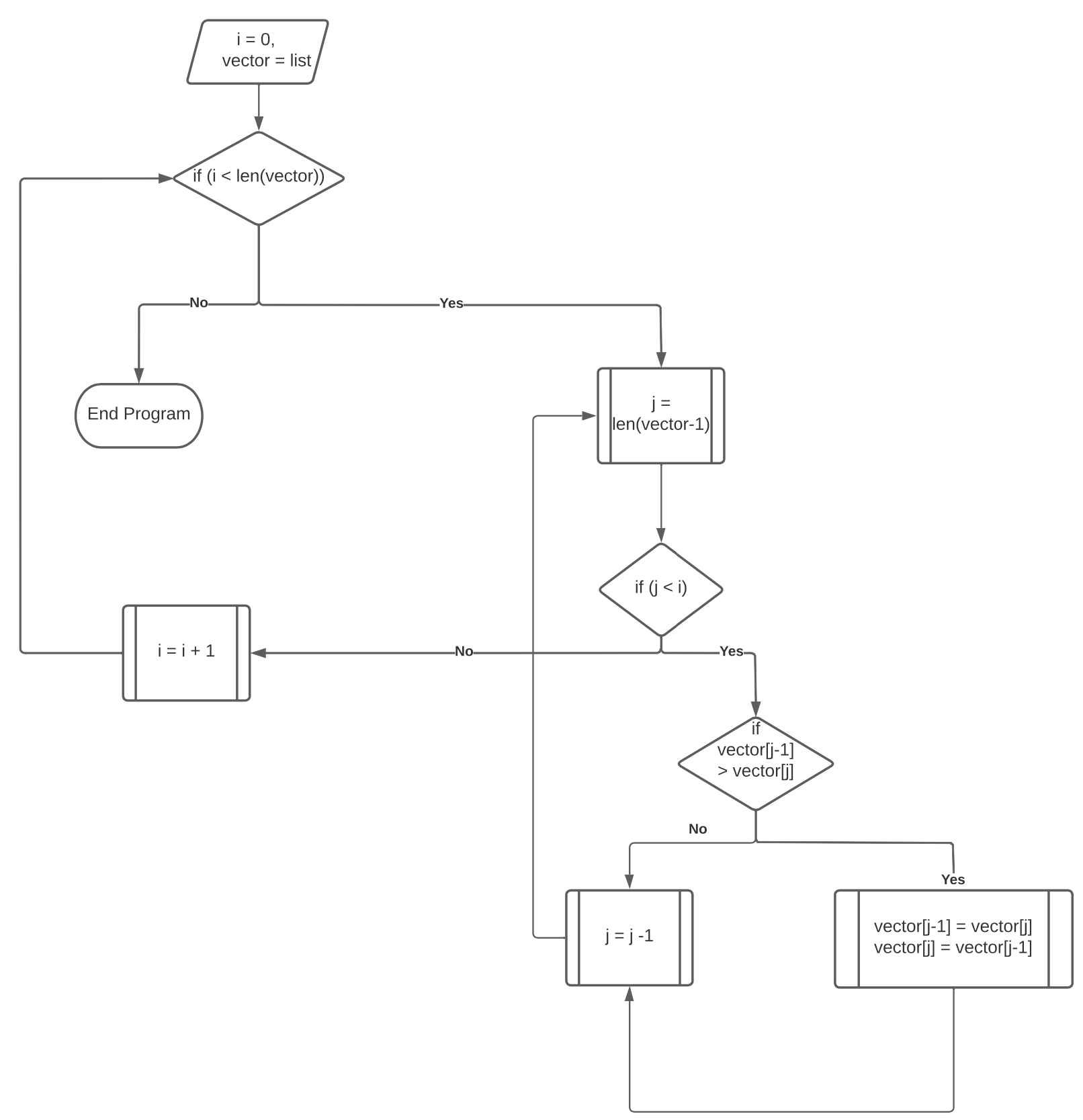
if array[b-1] > array[b]:

array[b-1] = array[b]

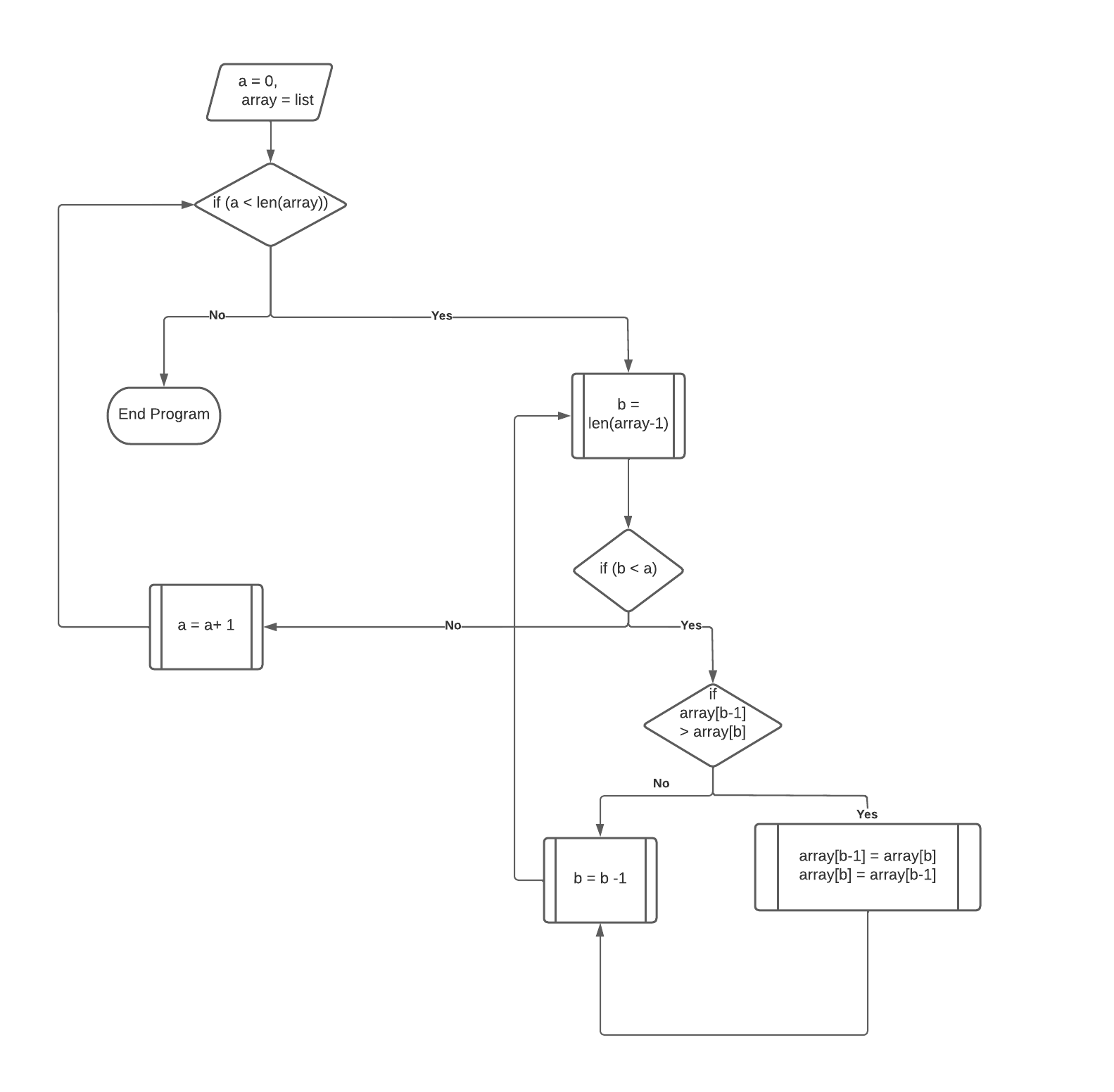
array[b] = array[b-1]

*fig 1b: Another implementation of the Bubble Sort algorithm*

This difference means that to a neural network trained on vectors, they are two completely different pieces of text. On the other hand, when represented by a tree or flowgraph, as in fig 2a and fig 2b, it can be seen that both programs have the same tree and graph representations.



*fig 2a: Flowgraph representation of code snippet from fig 1a*



*fig 2b: Flowgraph representation of code snippet from fig 1b*

It provides the background of the work. The problems and project objectives should be stated comprehensively. The motivations of the project should be presented. The techniques and approaches used to deal with the problem should be stated with justifications, and the contributions and main results achieved should be stated clearly. The structure of the report can be described briefly at the end.

## Background and Literature Review

While the field of Programming Language Understanding is one with little pre-existing literature, there has been some interesting work done in building neural networks specifically for processing source code. Objective 1 from section 3.1 is loosely based on the methods described in a paper that explores Learning to Represent Programs with Graphs [\_\_]. In this paper, the researchers define a method for converting source code to graphs based on the AST [\_\_] representation of the source code. To the best of my knowledge, this paper is the first of its kind which makes it very relevant to the work done in this project. This paper also makes use of the Gated Graph Neural Network (GGNN) to carry out two learning tasks on source code. These tasks are: 1). VARNAMING which is designed to predict what a variable should be called based on how it is used in the program and 2). VARMISUSE in which the model predicts whether a variable has been used correctly or not.

On the VARNAMING task, they receive an accuracy of 53.6% while on the VARMISUSE task, they receive an accuracy score of 85.5%. The results of the VARNAMING task indicate that the model used in this study might need modifications to its design to produce stronger results on similar tasks. On the other hand, the results of the VARMISUSE task indicate that the GGNN model can, to a high level of accuracy, tell when a variable has been misused. These results are a motivation for this project. They indicate that neural networks can be used to process and understand source code, but the model has to be specifically designed to handle data types that accurately represent the complex information contained in source code.

Objective 4 from section 3.1 is also based on work done by researchers in the past [\_\_]. This paper presents a method for processing source code by introducing the Tree-Based Convolutional Neural Network (TBCNN). The method in this paper makes use of convolutional layers and dynamic pooling to process a vector representation of a program’s AST. It is also, to the best of my knowledge, the first of its kind. The most interesting aspect of this paper is its use of convolution. Convolution is commonly used when processing images [\_, \_] and the researchers admit that Convolutional Neural Networks (CNNs) [\_, \_] do not accurately represent the tree-based information. The results from this study show – a classification accuracy of 94% on a task to classify programs by functionalities – that with the right techniques, conventional neural network models that are usually unsuitable for working with source code can be utilised and structured in such a way that they produce good results when trained and tested on source code.

Additionally, there has been interesting work done on training neural networks to detect syntax errors in programs [\_\_] and also on how to convert code to embeddings suitable for processing by a neural network [\_\_]. These studies also fall under the field of Programming Language Understanding.

## Motivations

A motivating factor for this project is the shortage of existing literature in the field of Programming Language Understanding. To the best of my knowledge, there have been no studies done comparing tree-based models to graph-based models to assess which is the more suitable model for carrying out classification tasks on source code.

Overall, the most important motivation for this project is to examine the way tree-based neural networks and graph-based neural networks function at the lowest level. I will investigate how they can be utilised for training neural networks to differentiate between text and source code and how they can be used to understand the underlying structure of a program.

It gives an overall picture about the work with a clear review of the relevant literature. The background of the project should be given. What have been done to deal with the problem should be stated clearly. The pros and cons of various existing algorithms and approaches should be stated as well. Differences between your proposed method and the existing ones should be briefly described. It is important to make sure that the discussion is structured and coherent; the key issues are summarised; key and relevant references are used critically analysed and the literature is covered comprehensively.

# Background Theories

The underlying theory behind this project is that using the right design and model specifications, a neural network can be trained to carry out classification tasks on source code, with a high level of accuracy.

Objective 2 of section 3.1.1 has its background in the fundamentals of the Graph Neural Network (GNN). This network architecture, as described in [\_\_] presents a method for processing information stored in the structure of a graph using a specially designed Neural Network. There have been several developments on this model, including the Gated Graph Neural Network [\_\_], the Gated Graph Sequence Neural Network [\_\_], the Gated Graph Convolutional Neural Network [\_\_] and the Gated Graph Recurrent Neural Network [\_\_], etc. With this project, I will build on the generic GNN to create two graph-based models that can accurately classify source code files.

The background of Objective 5 in section 3.1.1 is Natural Language Processing [\_\_]. This is a field of Artificial Intelligence with a lot of previous work done to understand how neural networks process natural text. The shortcoming of NLP models, specifically the models chosen for this project (the FFN and the LSTM) is that they convert each individual word or a series of consecutive words into a token which is then converted into a vector to be processed by the model. This vector representation is unsuitable for source code tasks.

This is because in natural language, each language has its own vocabulary. With programs, the vocabulary changes depending on who wrote the program. This means that all the files in a training set made of programs could have little to no words or tokens in common, leading to under-fitting in the model.

Another reason vector embeddings are unsuitable for program understanding is because in natural language, certain concepts do not exist, but they exist in source code. An example of this is a loop. In natural language, this would be the equivalent of saying the same thing over and over again (repetition of words or tokens), but in source code, you simply create a conditional statement to produce the desired output, removing the need to repeat the statement.

The general background of this project is rooted in the Graph data structure and the Tree data structure. I believe that there has not been enough work done to examine how these data structures can be used to represent the information contained in programs. This project will demonstrate how graph-based and tree-based neural network models are more suitable than text-based models for understanding and processing the complex information contained in source code.

The background theories supporting the work should be given in this section. Provide references when someone’s work is recalled.

# Objectives, Specifications and Design

## Specific Project Objectives

The main objective of this project is to compare graph-based neural network models to tree-based neural network models and compare both of these to text-based neural network models to determine which of these model types is the most suitable for carrying out classification tasks on source code based. I have further divided this main objective into 6 parts.

Objective 1: To implement a method of converting source code into a directed graph, containing nodes and edges. The nodes will be the elements in the program and the edges will be directed, showing the flow of the program.

Objective 2: To utilise the graph from Objective 1 to develop two graph-based neural network models; the Graph-Based Feed-Forward Neural Network (GFFN) and the Graph-Based LSTM Neural Network. These models will be capable of carrying out classification tasks on source code based on the categories laid out in section 3.1.1.

Objective 3: To implement a function for converting source code into a tree structure based on its AST representation. As with Objective 1, the nodes in the tree will represent the elements in the program and the edges will show the relationships between two nodes and the flow of the program.

Objective 4: This objective is based on Objective 3. This is to develop the Tree-Based Feed Forward Neural Network (TBFFN) and the Tree-Based LSTM Neural Network. These models will accept the trees implemented in Objective 3 as their input and carry out classification tasks on these trees based on the categories presented in section 3.1.1.

Objective 5: To develop two different text-based models to compare their results when trained and tested using source code. These models are the Long Short-Term Memory (LSTM) Neural Network and the Feed-Forward Neural Network (FFN).

Objective 6: This is to compare the performances of each of the 6 models described above in each of the categories defined in section 3.1.1 to determine which model is most suitable and accurate for carrying out classification tasks on source code.

### Classification Categories

For each of the six models outlined above, there are two classification tasks.

1. A binary classification task to differentiate between the Merge Sort and the Quick Sort.
2. A multi-label classification task to differentiate between the Merge Sort, the Quick Sort, and any other type of sorting algorithm.

For the two text-based models from Objective 5, I have included a third classification task to further demonstrate the shortcomings of text-based models when they are trained and tested on source code.

1. A multi-label classification task to differentiate between Python code, Java code and C code.

## Technical Project Specifications

### Models

The code for this project has been written using the Python programming language and a range of Python-specific deep learning libraries; Keras, TensorFlow, ETE Toolkit, NumPy, Scikit Learn, Matplotlib and NetworkX.

The Python standard library and TensorFlow have been used for majority of the implementations in this project. The graph-based models and the tree-based models have been written using a combination of standard Python and TensorFlow. Some Keras is used in these models but only to a small degree. All the variables are declared as tensors and all the calculations are carried out using TensorFlow.

The text-based LSTM model has been built entirely with Keras while the text-based FFN model has been built using standard Python and TensorFlow. In this model, the weights and biases are declared as TensorFlow variables and all the calculations are carried out on tensors. For both of these text-based models, the input files are parsed and embedded using NumPy and Scikit Learn. As a result, the text-based models will take only a series of NumPy vectors as their input for the training data. The training labels on the other hand, are categorical data converted from their original form into TensorFlow categorical variables.

### Datasets

All the datasets used in this project have been collected from GitHub. The datasets are divided into two parts, based on the categories in section 3.1.1. The first data set, based on the first two classification categories from section 3.1.1, is made up entirely of Python code, divided into three groups: Merge Sort, Quick Sort, and Other. This dataset is the main dataset of focus for this project. It is the dataset I will be using to train and test all 6 models.

The Merge Sort group is made up of 84 different implementations of the merge sort algorithm. The Quick Sort group contains 77 implementations of the quick sort algorithm and the Other group contains 71 implementations of different sorting algorithms, not including the quick sort and the merge sort.

The second dataset is based on the third classification category of section 3.1.1. It is also divided into three groups: C, Java and Python. The C group contains 350 programs written in C, the Java group contains 225 Java files and the Python group contains 160 Python files. This is a secondary dataset and it is only used in a multi-label classification task on the text-based models to further show the low levels of accuracy provided by text-based models when they are trained and tested on source code.

## Design

There are 6 different neural network models that have been implemented as part of this project. Two text-based models, two graph-based models and two tree-based models.

### Text-Based LSTM Model

The LSTM is a type of Recurrent Neural Network (RNN) that was developed to deal with the vanishing gradient problem [\_\_] that occurs in most other RNNs. It has since become a very popular model, and the most cited neural network model of the 20th century [\_\_].

The text-based LSTM model for this project is made up of an input layer (input), 2 hidden LSTM layers with 256 neurons each, 1 of which is bidirectional (lstm\_1) and one of which is uni-directional (lstm\_2), a dropout layer with a probability of 0.3 (dropout), and an output (dense).layer with the number of neurons depending on the classification task from section 3.1.1. It has been implemented using the Keras machine learning library [\_\_].

I have chosen to use a bidirectional layer [\_\_] in lstm\_1 because of the way bidirectional LSTMs work. With this type of layer, the input moves in two directions; input-to-lstm\_1 and lstm\_1 to input. This means that the information from both layers is preserved and the results are likely being passed to lstm\_2 are likely to be more accurate.

I have made lstm\_2 unidirectional to prevent overfitting – a situation where a neural network learns the training data too well and cannot perform well on unseen tasks. This layer passes information in one direction (lstm\_2-to-dropout).

The dropout layer has been inserted to drop 30% of the weights at random to override any overfitting that may have occurred in the previous layers. This is the final layer before the output layer.

This model has been built using Keras. It is the only model that was built using Keras. This is because Keras provides a lot of high-level functionality which is particularly important when working with a model as complex as the LSTM.

Before deciding on this selection of layers, I experimented with different combinations of layers and layer types. I chose the above because it is the combination that produced the highest accuracy of all my experiments.

### Text-Based FFN Model

This model has been built from scratch using TensorFlow and standard Python. It is made up of one input layer (input), two hidden layers (hidden\_1 and hidden\_2) with 128 neurons each, and an output layer (output). It applies back-propagation across all the layers to fine-tune the output of the model. The entire process is broken down below:

1. The forward pass is carried out across all the layers. The transfer function used in all the layers is the logarithmic sigmoid transfer function:

I have chosen this equation because out of all the transfer functions I experimented with, this equation produced the highest levels of accuracy.

1. The loss (the difference between the actual weight and the predicted weight) is calculated the TensorFlow cross entropy function.
2. The gradient of the weights and the loss is computed as the value *g*.
3. The value *g* is multiplied by the learning rate to give a value *i*.
4. The difference between the value *i* and the predicted weight is calculated to get a value *j*.
5. The value *j* is set as the final weight to get the output.

### Graph-Based Model

Input Layer

#### Graph-Based FFN Layers

#### Graph-Based LSTM Layers

### 

Input Layer

#### Tree-Based FFN Layers

#### Tree-Based LSTM Layers

It recalls the objectives in a more detailed way to justify the development of a set of requirements and specifications and identify a coherent set of issues to be addressed. It explains in detail the design and how the design can achieve the project aim (solve the problem).

# Methodology and Implementation

In total, I have constructed three identical LSTM models – one for each category defined in section 3.1.1.

It presents and justifies the methodology used to deal with the problem and describes in detail the implementation procedures. The background theory presented in the previous chapter can be recalled to support the proposed implementation. The originality, novelty and contribution are to be demonstrated with the discussion of the strengths and limitations.

# Results, Analysis and Evaluation

It summarises the results obtained from the proposed design and methodology. The way to obtain the results should be described in detail. Analysis and evaluation have to be performed. Comparisons should be made. It should justify if the project aims, objectives, requirements and specifications have been achieved.

# Legal, Social, Ethical and Professional Issues

A chapter gives a reasoned discussion about legal, social ethical and professional issues within the context of your project problem. You should also demonstrate that you are aware of the Code of Conduct \& Code of Good Practice issued by the British Computer Society (BSC) (https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct) for computer science project and Rule of Conduct issued by The Institution of Engineering and Technology (IET) (https://www.theiet.org/about/governance/rules-of-conduct) for engineering project. You should have applied their principles, where appropriate, as you carried out your project. You could consider aspects like: the effects of your project on the public well-being, security, software trustworthiness and risks, Intellectual Property and related issues, etc.

# Conclusion

It is a chapter to sum up the main points and findings of the work; how you achieve the project aims and address the research questions; the contributions and results you have achieved. Future plan and development can be mentioned in this section as well. It is normally in one or two pages.

# References

Refer to the citing reference information on KEATS

# Appendices

## Appendix A:

## Appendix B:

Supplementary materials (such as source code, user menu, etc) could be included. Each appendix must be labelled (for example, Appendix A, Appendix A.1, Appendix A.2, Appendix B, Appendix B.1, etc.) and with heading. All Appendices must be referred in the text.

## Appendix B: Points to Note

* Please note the following points when you write your report:
* Consider the outline of the report. It is a good idea to start with the table of contents, which gives you an overall structure of the report.
* Show understanding of the topic and demonstrate the contribution of the work. 70\% of the content of the report should be your own contributions and achievements.
* Always use your own words.
* The main report and any appendices must constitute one document.
* Pages must be numbered consecutively.
* Captions must be provided for all figures and tables.
* Equations (or important equations), figures and tables must be numbered.
* All figures and tables must be referred to in the text.
* Units of all variables must be provided.
* Numerical values (floating-point number) should be in 4 decimal places.
* Contractions should not be used.
* Check the punctuation of sentences. In particular, those sentences with equation. For example, if an equation is at the end of a sentence, a full stop should be used.
* All variables must be defined.
* Font face of variables throughout the report (in the text, equation, figures and table) must be consistent.
* Use proper headings for chapters, sections, subsections.
* Chapters, sections, subsections should be numbered and with the same numbering system throughout the report.
* It is suggested that vector and matrix variables should be in bold, scalar variables should be in italic.
* References must be used for materials used in the report that are not yours.
* A standard reference format must be adopted and be consistently applied through the report. General guidelines for reference format can be found on KEATS.
* Always backup your files.