Multiclass classification via Transformer Networks for Task Constraint-based Introductory Programming Feedback

Gerd Lowell Jana
Bachelor of Science in Computer Science

John Kenneth Lesaba
Bachelor of Science in Computer Science

Karl Frederick Roldan
Bachelor of Science in Computer Science

Senior project submitted to the faculty of the
Department of Computer Science
College of Computer Studies, Ateneo de Naga University
in partial fulfillment of the requirements for their respective
Bachelor of Science degrees

Project Advisor: Joshua C. Martinez, MIT
Raphael Henry M. Garay
Jelly P. Aureus, MS
John Sixto G. Santos, MS

May, 2022
Naga City, Philippines

Keywords: CS Education, transformer

The Senior Project entitled

Multiclass classification via Transformer Networks for Task Constraint-based Introductory Programming Feedback

developed by

Gerd Lowell Jana

Bachelor of Science in Computer Science

John Kenneth Lesaba

Bachelor of Science in Computer Science

Karl Frederick Roldan

Bachelor of Science in Computer Science

and submitted in partial fulfillment of the requirements of their respective Bachelor of Science degrees has been rigorously examined and recommended for approval and acceptance.

Raphael Henry M. Garay
Panel Member
Date signed: ________________________

Jelly P. Aureus, MS
Panel Member
Date signed: ________________________

John Sixto G. Santos, MS
Panel Member
Date signed: ________________________

Joshua C. Martinez, MIT
Project Advisor
Date signed: ________________________

The Senior Project entitled

Multiclass classification via Transformer Networks for Task Constraint-based Introductory Programming Feedback

developed by

Gerd Lowell Jana

Bachelor of Science in Computer Science

John Kenneth Lesaba

Bachelor of Science in Computer Science

Karl Frederick Roldan

Bachelor of Science in Computer Science

and submitted in partial fulfillment of the requirements of their respective Bachelor of Science degrees is hereby approved and accepted by the Department of Computer Science, College of Computer Studies, Ateneo de Naga University.

Adrian Leo Pajarillo, MS
Chair, Department of Computer Science
Date signed: ______________________

Joshua C. Martinez, MIT
Dean, College of Computer Studies
Date signed: ______________________

Declaration of Original Work

We declare that the Senior Project entitled

Multiclass classification via Transformer Networks for Task Constraint-based Introductory Programming Feedback

which we submitted to the faculty of the

Department of Computer Science, Ateneo de Naga University

is our own work. To the best of our knowledge, it does not contain materials published or written by another person, except where due citation and acknowledgement is made in our senior project documentation. The contributions of other people whom we worked with to complete this senior project are explicitly cited and acknowledged in our senior project documentation.

We also declare that the intellectual content of this senior project is the product of our own work. We conceptualized, designed, encoded, and debugged the source code of the core programs in our senior project. The source code of third party APIs and library functions used in my program are explicitly cited and acknowledged in our senior project documentation. Also duly acknowledged are the assistance of others in minor details of editing and reproduction of the documentation.

In our honor, we declare that we did not pass off as our own the work done by another person. We are the only persons who encoded the source code of our software. We understand that we may get a failing mark if the source code of our program is in fact the work of another person.

Gerd Lowell Jana
4 - Bachelor of Science in Computer Science

John Kenneth Lesaba
4 - Bachelor of Science in Computer Science

Karl Frederick Roldan
4 - Bachelor of Science in Computer Science

This declaration is witnessed by:

Joshua C. Martinez, MIT

Project Advisor

Multiclass classification via Transformer Networks for Task Constraint-based Introductory Programming Feedback

by

Gerd Lowell Jana, John Kenneth Lesaba, and Karl Frederick Roldan

Project Advisor: Joshua C. Martinez, MIT

Department of Computer Science

(ABSTRACT)

Task constraint feedback is a kind of feedback that checks whether problem-defined constraints were fulfilled by students upon submission of work. The feedback process can be as simple as checking if a certain programming construct exists or if the student used a specific algorithm or data structure as prescribed by the problem. Most task constraint systems, such as *WeBWoRK-JAG* and *Fischer06*, use basic static analysis and comprehensive automated testing. ProPL, a task constraint checker, is an exemption which uses natural language processing techniques to generate feedback. In this study, the proponents will use Transformer Networks which have been known to work really well with sequence-to-sequence natural language models such as neural translation, text generation, and summarization. Previous works showed evidence that transformer networks can be generalized for use in programming language processing such as code generation. The study will apply transformer networks to generate classes that are used in the submitted codes of students to generate relevant task constraint feedback. This study aims to show how transformers can be applied in aiding computer science education through task-constraint feedback.

I dedicate this research work to all of humanity.

ACKNOWLEDGEMENTS

I thank everyone who helped me finish this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Programming is the norm in virtually every STEM field such as natural sciences, mathematics, engineering, and statistics. Due to this, studying fundamental concepts such as data structures and algorithms are needed for programmers as core knowledge. Since competitive programming is one of the main ways to introduce these concepts, the assessments of these programs enable the skills of a programmer to develop[42].

Automated assessment of programming codes has been widely used in competitive programming sites such as HackerEarth, CodeChef, etc. to aid in evaluating solutions to problem sets. Most of these systems utilize black-box testing wherein it inputs test cases to the submitted solution, and compares its output to the expected output encoded by the problem setter. In this paper, we aim to build on existing automated program evaluation methods by adding algorithm detection in order to allow problem setters to fine-tune their problems by means of assigning and implementing task-constraints, and in turn provide a more formative feedback to students. The researchers will use the current state of the art Transformer Network to act as a classifier. This feedback system will then be implemented on Project CodeC (Coder's Circle), a multifunctional, web-based platform that aims to combine competitive programming, social networks, and a classroom-based Virtual Learning Environment (VLE), and is currently under the development of Ateneo de Naga University, College of Computer Studies.

1.1 Project Context

In task-constraint based feedback, an additional verification of student submission is done on the submission checker whether the student was able to complete the task. On a control structure level, this is easy since the task-constraint can be observed statically using a parser and checking whether the code uses an 'if' or a 'while', etc.

This problem is more difficult when the task is to use a certain algorithm. For example, a teacher may require the students to use the quicksort algorithm in the problem instead of merge sort. However, when a student submits a code to an automated checker, whether quicksort or merge sort will be unknown to the compiler due to the similar time complexities and may still pass the problem time and space constraints.

Project CodeC aims to implement a task-constraint feedback system that fails a submission if the submitter hasn't been able to complete the task set by the problem setter despite passing the space and time constraints of the problem. A sample feedback might be along the lines of "Oops! Are you sure you have implemented a hashmap?"

1.2 Purpose and Description

Aside from checking the submitted program's correctness using traditional code-checking, Project CodeC aims to give automated formative feedback to ensure that students are learning through programming. One of these feedbacks is the task-constraint feedback. A problem setter may lay out some task requirements for the problem, such as whether to implement and use that specific requirement, which is useful in introductory computer science education.

The researchers propose the use of algorithm classification on checking whether task-requirements are fulfilled during the code submission. To do this, the problem setter denotes some task constraints which could be specific algorithms that he/she wants implemented in the student solution. When a student passes all the time and space constraints of the submission, the submission is then checked if it contains all task requirements that the problem setter had chosen. The feedback part notes the tasks required but were not implemented and tells the student that they have not implemented the said tasks.

In this paper, we will answer the following research questions:

RQ1. How effective would the CodeBERT Transformer model be in an algorithm classification task?
RQ2. Which algorithms does the Transformer perform best/worst on?
RQ3. How would changing the loss function affect the performance of the algorithm detection model?

While the research aims to classify code snippets into which algorithms they implement, which is a subset of code summarization, the researchers believe that no research on algorithm classification has been done yet.

1.3 Objectives

General Objective

- This research aims to develop an algorithm classification machine learning model.

Specific Objectives

- Classify algorithms given code snippets mined in Github reporsitories using a multilabel classification algorithm.
- Study how the model will perform as a task-constraint feedback system using the CodeChef code submissions dataset.
- Deploy the machine learning model to Project CodeC as a service if it is deemed as accurate enough for real-world use.

1.4 Scope and Limitations

Due to the limitations on covered programming languages by CodeBERT, only Java, Python, and JavaScript programs will be considered. Furthermore, due to the unavailability of an already-existing dataset, algorithms considered are constrained to linear search, binary search, some sorting algorithms, linked lists, and hash maps, which are discussed in any algorithms classes.

Also, since the research only covers the classification of algorithms. It does not aim to find if an algorithm is implemented using recursion or a for loop as that can be done using static analysis as shown in Chapter 2. Another area not discussed in this paper will be the effectiveness of task-constraint feedback systems in programming assignments for computer science education.

Chapter 2

Review of Related Literature

Competitive Programming (CP) is a sport that involves programmers competing against each other to solve programming problems with a time limit. The programmers are then evaluated based on the factors such as efficiency, time complexity, and the accuracy of the code.[37] This paradigm was based on an event in Bulgaria 30 years ago called the International Olympiad in Information (IOI). Nowadays, the majority of these competitions are done entirely online using platforms such as CodeChef, HackerRank, HackerEarth, and others. It can have contestants ranging from two to several thousand and span topics such as data structures & algorithms, data science, and discrete mathematics[40].

2.1 Competitive Programming as Pedagogy

The National University of Singapore, University of West Indies, and other universities have been using CP as an alternative method of teaching students by utilizing it as a module in replacement to intermediate-level data structures and algorithms subjects[8]. Training programs[11] and teaching frameworks[12] were also created revolving around CP as the major learning material which uses Association for Computing Machinery International Collegiate Programming Contest (ACM-ICPC) and IOI style of evaluating the performance of the students. Furthermore, employers and graduates are utilizing CP as a preparatory material to adapt to the competitive environment of the software industry.[40] Nair states that in India, many people joining the IT workforce are unemployable and use CP to increase employability and introducing CP not only improves the skills of these students but also dramatically enhances their capability of collaboration and creativity.

2.2 Evaluation of Programming Codes

Most competitive programming sites today use automatic assessment of programming codes that has two primary approaches[21]: *Static analysis*, which is done by inspecting the source code of the program without executing it; and *Dynamic analysis*, wherein a program is run and tested against different test cases.

2.2.1 Existing methods of automated evaluation of programming codes

Competitive programming web applications use numerous methods[21][41] in evaluating programming code. Two methods have been proposed[41] to evaluate codes:

1. The Association for Computing Machinery (ACM) - style implements black-box testing then identifies the runtime complexity and checks if it meets the maximum runtime and memory limits set by the problem setter. Only the solutions that meet these requirements are given credits and the most runtime efficient of them are ranked the highest. This method is used in ACM ICPC, CodeChef Cook - off contest, etc.; and

2. The International Olympiad in Informatics (IOI) - style evaluation first groups the test files according to its asymptotic complexity (e.g. $O(n)$, $O(n \log n)$, etc.) or logical complexity(e.g. problem restricted to an easier subproblem) and are given point equivalents. The judge gives the partial points to solutions that have the perfect (equal) complexity as it is tested among the grouped test cases. This method is used by IOI, CodeChef Lunchtime, etc.

Furthermore, numerous techniques[22][53][36][57] were proposed in assessing programs that may further the progress in computing education.

2.2.2 Task-constrained feedback in automated assessment systems

Task-constraint feedback systems are programs that automatically give feedback on whether the submitted solution implements the required task. Among these systems include WeBWork[18] which focuses on error correction, INCOM[33] which acts as a homework assistance system. The feedback from these systems are considered essential, especially in developing countries where resources and student-professor contact hours are limited.

Automated assessment systems perform tests that mimic that of the Software Quality Assurance (SQA) testing[18]. These tests are done for validation that the program does what it is intended to do, and for verification that the program works correctly through black-box testing. Gotel et al. proposed a framework that incorporates these SQA principles in assessing programming code. ProPL[31] introduced a similar approach in identifying program goals and schemas, as well as in implementing correct solutions to problems.

2.3 Sequence Models

Sequence models are machine learning algorithms that are designed to input and/or output sequential data such as time series, audio, signals, texts, and videos. Four common sequence model types have been proposed: one-to-many, many-to-one, many-to-many with equal input and output lengths, and many-to-many with varying input and output lengths. Such models were used in applications including translation, image captioning, sentiment analysis, and many more[58].

Different neural networks had been proposed with varying performance on different tasks in the past decades. The Recurrent Neural Network[23] was one of the first sequence models that was introduced. The LSTM[25] and the GRU[6] were introduced as modified versions of the hidden layer of RNN which showed significant performance improvements and eliminated the vanishing gradient problem of the RNN. Vaswani et al.[51] had taken a different approach by introducing a model, the transformer, that takes a sequence as a whole instead of sequentially (as in the RNN and its variants) which renders NLP to be easily parallelized and self-attention which computes how words in a sequence are related to each other.

2.3.1 Advances on Programing Language Processing in Machine Learning

Recent breakthroughs have shown that sequence models can be generalized to programming languages as well. Central themes on research include code summarization[56] which can help in automatically commenting a snippet of code, code-clone detection[54] which can be used for finding functional similarities for two code snippets that look distinct from each other, and code completion[3] which can be used for assistance when coding.

Processing programming languages on a neural network implies that code has to be embedded to a series of vectors that represent code semantics or syntax which can serve as input by the network. Due to this, much work has been done on developing ways to generate code embeddings such as processing Abstract Syntax Trees (AST) on CNNs and RNNs,[39, 5] descriptive program embeddings of a single code snippet on a function level[2] by encoding through AST paths[1], and using Hoare triples of a program[44].

Chapter 3

Theoretical Background

3.1 Transformers

Vaswani et al. [51] described transformers as sequence models that solely focus on self-attention without the need for recurrence which eliminates the need for unfolding during the backpropagation phase of learning. In the paper, attention was defined with respect to keys, values, and queries. The attention is computed with

$$\text{attention}(K, Q, V) = \left(\frac{QK^T}{\sqrt{\dim(K)}}\right) V$$

where K is the key matrix, Q is the query matrix, and V is the value matrix. The dot product of Q and K is scaled by the square root of the dimension of K.

Transformer models take the whole input as a whole instead of recurrently meaning that the model does not know the order of the words in the sequence. Hence, the paper proposed the use of positional encodings through sinusoidal functions added to the embedded vectors. Suppose that W is a matrix where each column corresponds to a position *pos* and each row *i* corresponds to the individual dimensions in the model which alters the frequency. Let $W = 10000^2 i\text{embedsize}$ where embedsize is a hyperparameter which is also the embedding size of the embedded vectors. Then, the positional encoding is calculated by

$$POS(pos, i) = \sin\left(\frac{pos}{W}\right) \text{ if } i \text{ is even and}$$

$$POS(pos, i) = \cos\left(\frac{pos}{W}\right) \text{ if } i \text{ is odd}$$

The key component of transformers is the multi-headed attention layer, wherein attention is computed multiple times and concatenated together. Transformers were able to outperform established models for machine translation with English-German translation.

3.2 RoBERTa and CodeBERT

RoBERTa[35] is an auto-encoding transformer model that was a replication study of the original BERT model by finer-tuning hyperparameters and removing the Next Sentence Prediction task.

3.2.1 Model Architecture

RoBERTa is an auto-encoder transformer with an almost identical architecture as the original transformer [51]. The model, however, uses a bi-directional multi-headed self-attention which will be discussed further in the next subsection. This can be done using Masked Language Modeling (MLM). Like BERT, RoBERTa was trained in two states: pre-training and fine-tuning.

3.2.2 Pre-training

Since sequence models can only be trained left-to-right and transformers take code non-sequentially, BERT used the Masked Language Modeling (MLM) task to achieve bidirectionality.[10] To achieve this, the BERT authors masked 15% of the words in an input sentence and had BERT predict which word it should be. This achieved a sense of bidirectionality wherein the sentence was able to see itself. Furthermore, BERT was pretrained on an NSP task which attempts to predict the relationship between two sentences.

Unlike BERT, RoBERTa was pre-trained on a larger corpus of 160GB of English data and removed the NSP task as it was shown that it does not affect the performance of the model.

While BERT masked the MLM pretraining inputs during data preprocessing, RoBERTa, however, masked the input while feeding data into the encoder which was found to be useful for larger datasets[35].

3.2.3 Fine-tuning and Feature Extraction

Fine-tuning refers to retraining a transformer with a custom and smaller dataset to adjust the hyperparameters to the specific dataset and task it was given. In RoBERTa, the pre-training step is done to learn language representations and fine-tuning would be needed for downstream tasks such as classification and named entity recognition. Fine-tuning is less expensive than training from scratch and can be done by adding a single layer for classification.

It is also possible to use RoBERTa as a feature extractor for an existing machine learning model, such as an SVM or a deep neural network, by using RoBERTa to compress the input and extract the learned features. The model's gradients can be frozen for feature extraction, that is, there is no retraining of the model.

3.2.4 CodeBERT

In this study, the researchers will use the CodeBERT[15] auto-encoder model. The CodeBERT model is an exact replica of RoBERTa without any changes to its architecture, except that it was re-trained on a different dataset, namely, that of programming languages.

CodeBERT was pre-trained on a large corpus of about 8 million code snippets of both bimodal and unimodal data. Bimodal data means that for each code snippet, there is an accompanying natural language snippet while unimodal implies the absence of natural language snippets. The data were gathered from GitHub, which was collected by Husain et al[26]. CodeBERT does not accept code written in Assembly language and due to the constraints mentioned in section 2.3.1, CodeBERT has only been trained using code snippets written in java, javascript, and python, the researchers will not be able to use assembly language representation of code to reduce language-embedded noise.

3.3 Task-Constraint Feedback

Knowledge About Task-Constraints (KTC) feedback systems employ the use of limiting the submission by adding requirements, called *constraints*, that will tell the student if they haven't completed the requirement.[27] One of the methods used is *Hints on Task Requirements* (TR) wherein a hint will be given when a task was not completed. As an example, suppose that the problem setter asked to balance a binary search tree using red-black trees. However, the submitter was able to complete the problem using an AVL tree. The TR system should give some feedback that the Red-Black Tree is a requirement for the given problem.

3.4 Algorithm Classification

The researchers define algorithm classification as to what algorithm the code snippet is implementing, rather than a broad definition like sorting. Algorithms and data structures have multiple methods of implementation, and these methods which will act as labels in the classifier will be referenced through Introductory algorithm textbooks[52, 47, 46]. As an example, consider the task of implementing a *hash map.* This data structure can be implemented using *binary trees* or *linked list*; in this example, assume that it is implemented using *linked list*, therefore, it is not enough to simply classify the submission as a *hash map* since a *linked list* is also implemented.

Supposed that the code snippet implements an algorithm or data structure using the language's standard library, then the snippet will only be classified using the manual implementation disregarding the standard library function. Likewise, code snippets that do not belong to any algorithms stated in any of the introductory algorithm textbooks are said not to use any algorithm.

3.5 Focal Loss

Object detection algorithms are tasked with finding the foreground data while the background data are objects or signals in the input data that are not part of the target. Most state-of-the-art object detectors are two-staged. That is, finding the foreground parts and then classifying them. The focal loss [34] is an extension of the binary cross-entropy loss that aims to improve the performance of one-stage detectors (models that find foreground and classifications in one pass) such as the model used in this research.

$$\ell_{CE}(\widehat{y}, y) = \begin{cases} \log(\widehat{y}) & \text{if } y = 1 \\ \log(1 - \widehat{y}) & \text{otherwise} \end{cases} \tag{3.1}$$

with $y \in \{-1, 1\}$ which represents the target and $\widehat{y} \in [-1, 1]$ which represents the predicted probability. Following the original paper's notational convenience, define

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases} \tag{3.2}$$

and so $\ell_{CE}(\widehat{y}, y) = \ell_{CE}(p_t) = -\log(p_t)$. When there is a class imbalance in the dataset, the cross-entropy loss can be weighted by $\alpha \in [0, 1]$. For convenience, α_t is defined the same way as equation 3.2. Therefore,

$$\ell_{CE}(p_t) = -\alpha_t \log(p_t) \tag{3.3}$$

While α is a weighing factor for balancing an imbalanced dataset, it does not account for examples which are harder to classify. To do this, Lin, et al. proposed the focal loss function which introduces a modulating factor $(1 - p_t)^\gamma$ with $\gamma \geq 0$ as a *focusing* hyperparameter. Then, the focal loss is defined to be

$$\ell_{FL}(p_t) = -\alpha_t (1 - p_t)^\gamma \log(p_t). \tag{3.4}$$

While Yin, et al. tested focal loss on RetinaNet, a CNN-based object detection model, Cadavid et al. had tested the focal loss on transformer networks on maintenance logs which are classified as sequential data. [50]

Chapter 4

Methodology

4.1 Dataset

The researchers gathered a custom dataset mined from public GitHub repositories that contain Python, JavaScript, and Java source code. These repositories implement algorithms and data structures. Labels will be extracted from a number of heuristics such as method name or the repository folder in the case of algorithm implementation datasets. Since CodeBERT only accepts input lengths of less than 512, it is important to preprocess the dataset to make sure that each code snippet contains less than 512 tokens such as by removing extra newlines and comments[10][35].

The repository in [14] has an organized structure in contrast with repository [1]. Furthermore, the last repository only implements four algorithms in javascript and contains no comments that are descriptive of what the code does. Descriptive comments are present in the two former repositories and source codes are sorted into folders named after the algorithm implementations. Another notable difference between these examples is that in the first repository, there are no collaborators while in the second and third example there are multiple collaborators.

kadikraman/sorting[1]	iiitv/algos[14]	subbarayudu-j/TheAlgorithms-Python[113]
Source codes are named after the algorithm it implements	Source codes are organized into folders named after the algorithm it implements.	Algorithms are sorted into folders that are named after the general topic that corresponds to the algorithm (i. e. program codes that implement sorting algorithms are put inside a folder named "sorts"). Filenames for source codes are the specific algorithm it implements.
Codes are written in javascript.	Uses different programming languages	Uses Python only
Source codes are intended to be used as modules.	Source codes contain main functions and can be run standalone.	
Codes does not contain relevant comments that describe its code snippets.	Source codes include comments that describe its code snippets.	
Only implements four sorting algorithms covered by the study	Implements seven out of nine algorithms that are considered. Data structures (Hashmap and Linked List) were not included.	

Table 4.1: Github Repositories Structures

4.1.1 Dataset Analysis

The dataset was collected from almost five hundred GitHub repositories.

The distribution of programming languages in the dataset is unbalanced. There are 854 Java files, 758 Python files, and 226 Javascript files that were collected. There are in total 224 quicksort, 230 merge sort, 176 selection sorts, 226 insertion sorts, 223 bubble sorts, 253 linear searches, 107 binary searches, 208 linked lists, and 143 hashmaps implementations.

4.1.2 Dataset Preparation

Each GitHub file will be downloaded to a raw dataset folder. The preparation procedure will be as follows:

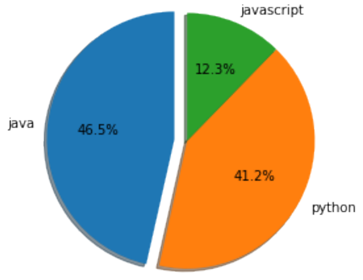


Figure 4.1: Total Number of Files per Programming Language

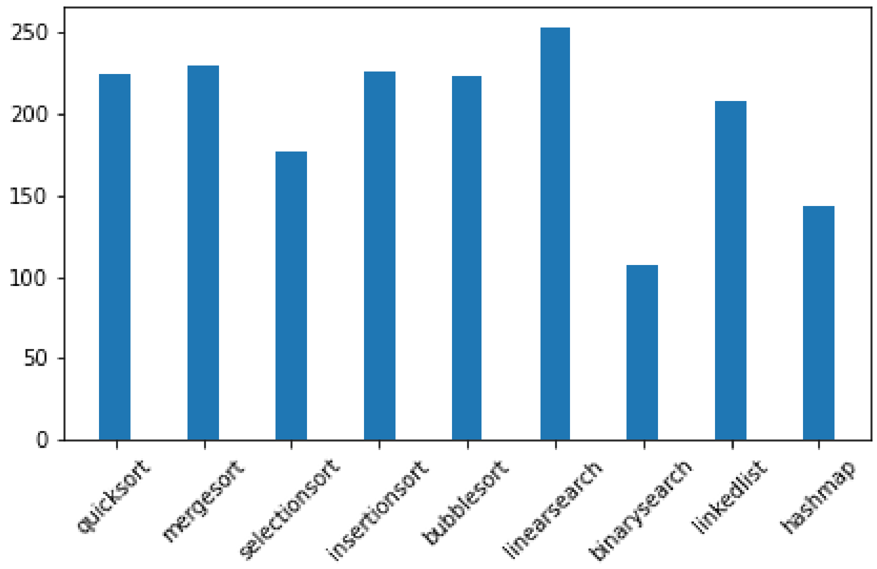


Figure 4.2: Total number of Labels

- Each file will be quickly scanned to see whether the file implements one of the target algorithms. Criteria for scanning includes filename, class names, folder names, function names, general code structure, or even the git repository name.

- Files written in Java, Python, or JavaScript that may implement one of the target algorithms will be put into a single folder. Filename clashes will be resolved by manually renaming and appending a number. (i. e. two files that are named bubbleSort.java, one file will be renamed as bubbleSort (1).java)

- An excel file that contains 10 columns, one for the filename, and nine for each target label, will be created. The data will be labeled following the data labeling protocol in Section 4.1.2.
- Comments and extra whitespaces will be removed in each of the files once labeling is completed so the neural network will not be influenced and decreased data size. This will be done through a script.
- Any duplicate data will be removed by calculating the similarity ratio of a file to other remaining files.

4.1.3 Dataset Biases

Dataset biases that are being expected by the researchers include

- Biases against specific implementation. To illustrate, there might be more recursive algorithms in some cases, and in other cases, there might be more imperative implementations. Whenever this happens, the machine learning model might favor one implementation of an algorithm over another alternative implementation. For example, suppose that the dataset has more recursive implementations of the quicksort algorithm with a ratio of 10:1 against the imperative implementation. In this case, the model might not consider quicksort inputs as quicksort.
- Biases against specific programming languages. Since the research is working on three different programming languages and the distribution of each language is not uniform, the model may favor one language over the other.As the distribution of languages is not uniform, the majority of the data points will be written in one language which may lead to misrepresentation of the model to the other languages considered. For example, suppose that the number of data points written in java, javascript, and python are at a ratio of 10:5:1. In this case, the model might not be able to transform python coded algorithms as well as it can java and javascript coded algorithms.
- Biases towards specific identifier names.

So far, the only real way of addressing such biases, without having to decrease the number of data, is to collect an even larger number of data for each programming language. However, this might not be possible for some algorithms due to the limited number of beginner programmers attempting to implement each algorithm in each language.

4.1.4 Data Labeling Protocol

The task of the model is a subset of code-summarization, specifically, determining whether a code input contains any of the nine algorithm targets mentioned in section 1.1.4. Most code-summarization data labeling processes use automated labeling under the assumption that function labels are often descriptive. However, that is not always the case. Take for example, the code snippet below:

```python
def f(n):
    return f(n - 1) + f(n - 2)

def f1(n):
    t = [1, 1]

    for j in range(2, n + 1):
        x = t[j - 1] + t[j - 2]
        t.append(x)

    return t[n]
```

In the snippet, both functions implement a fibonacci function, albeit different implementation details. However, the programmers in small single-file programs (the dataset the research uses consist of these) sometimes do not implement descriptive function names. With this, the researchers believe that manual labeling is the best option for this thesis.

Labels are to be extracted as follows

- The filename is the name of the algorithm or data structure. This is often a sign that it implements the code.

- The filename is the name of a data structure or an algorithm that may be implemented using one of the nine data structures in this thesis. For example, the file may be implementing a bucket sort and since bucket sort implementations often require the use of insertion sort, the file will be checked for insertion sort instead.
- The function name is descriptive and tells which algorithm it is implementing
- The implementation details look like the pseudocode listed in this paper.

Unlike other datasets such as sentiment analysis whose labels are subjective [4], algorithm recognition is an objective label. With this, the researchers aim for correctness by manually checking every datapoint to ensure that it implements the algorithm regardless of how it was implemented. When the label has a TRUE for every algorithm that the file includes and FALSE for every algorithm that the file does not implement, then the label is correct. Therefore, the researchers will be using a cross-walk labeling technique wherein labels are evaluated and labeled thrice: the researcher, the other researchers, and a field expert.

- All three researchers did individual labeling in the data set and recorded their labels in excel files.
- A script was written to find files with different labels made by each researcher.
- A worksheet was created as a team containing individual data labels per row.
    1. The worksheet contains 6 columns: 1 for the date that the row was reviewed, 1 for the filename, 3 for the labels of each of the 3 members, and 1 for the final label.
    2. The final labels are decided by:
        (1) If all of three researchers have the same labels on a data point, the label is considered final.
        (2) If one or more researcher/s have different label/s on a data point, the team consolidates to determine the final label.
- The inter-rater reliability is computed to ensure the consistency of the implementation of the rating system that is found in A.

1. If the inter-rater reliability is low, then an expert (i.e, the thesis advisor) verifies the labels that have been consolidated.

Below is an example of a piece of code and how it would be labeled:

```java
public class LinkedList {
    class Node {
        public int key;
        public Node next;

        public Node(int key) { this.key = key; }
    }

    public Node head;
    public Node tail;

    public void push(){};   /************************/
    public void delete(){}; /* Dictionary functions */
    public void find(){};   /************************/

    public int linearsearch(int key) {
        Node tmp = this.head;
        idx = 1;

        while (tmp != null) {
            if (tmp.key == j) {
                return idx;
            } else {
                idx++;
            }
        }
        return idx;
28      }
29  }
```

In the code shown above, it is evident that lines 1 to 14 implement a Linked List and its dictionary functions `push`, `delete`, and `find` while lines 16 to 28 implement a Linear Search algorithm on the linked list.

Hence, the data labelers should mark both *linear search* and *linked list* as `1` on the excel sheets. Here's an example of a label for a file:

Date Labeled	File Name	Linear Search	Binary Search	Insertion Sort	Bubble Sort	Quick Sort	Merge Sort	Selection Sort	Linked List	Hash Map
2/01/ 2022	file.js	1	0	0	0	0	0	0	1	0

Table 4.2: Label example

Below is an example of the worksheet that will be used to finalize the labels:

Date reviewed	filename	Member 1	Member 2	Member 3	Final
2/01/2022	file.js	lsearch; linked list	lsearch	lsearch	lsearch; linked list

Table 4.3: Final worksheet label example

4.1.5 Inter-rater Reliability

Inter-rater reliability [32] is the measurement of agreement among two or more raters and can be evaluated by a number of different statistics. It addresses the issue of consistency of the implementation of a rating system. The Intraclass Kappa Coefficient[29] is a statistical measure that assesses the level of agreement or reliability of multiple independent measures of the same unordered categorical form. Since data points in this research may be rated in more than one category, the researchers made use of the Extension of the kappa coefficient[30] in assessing the agreement among the raters that could select multiple categories for each object of measurement.

For the following, let N be the total number of data points, n be the number of raters, i and j corresponds to the item and category respectively, and k be the total number of categories.

$$P_i = \frac{1}{n(n-1)}\left[\left(\sum_{j=1}^{k} n_{ij}^2\right) - (n)\right] \quad (4.1) \qquad p_j = \frac{1}{Nn}\sum_{i=1}^{N} n_{ij} \quad (4.2)$$

Where P_i is the proportion of agreement in the classification of each data point and p_j is the proportion of observation under each category.

$$\bar{P} = \frac{1}{N}\sum_{i=1}^{N} P_i \quad (4.3) \qquad P_e = \sum_{j=1}^{k} p_j^2 \quad (4.4)$$

Then, $\bar{P}$ is the sum of all proportions of agreements per object divided by the total number of objects that were classified. P_e is used to calculate the random assignment of scores under each data point to take agreement by chance into account and is computed by the sum of the squares of each p_j.

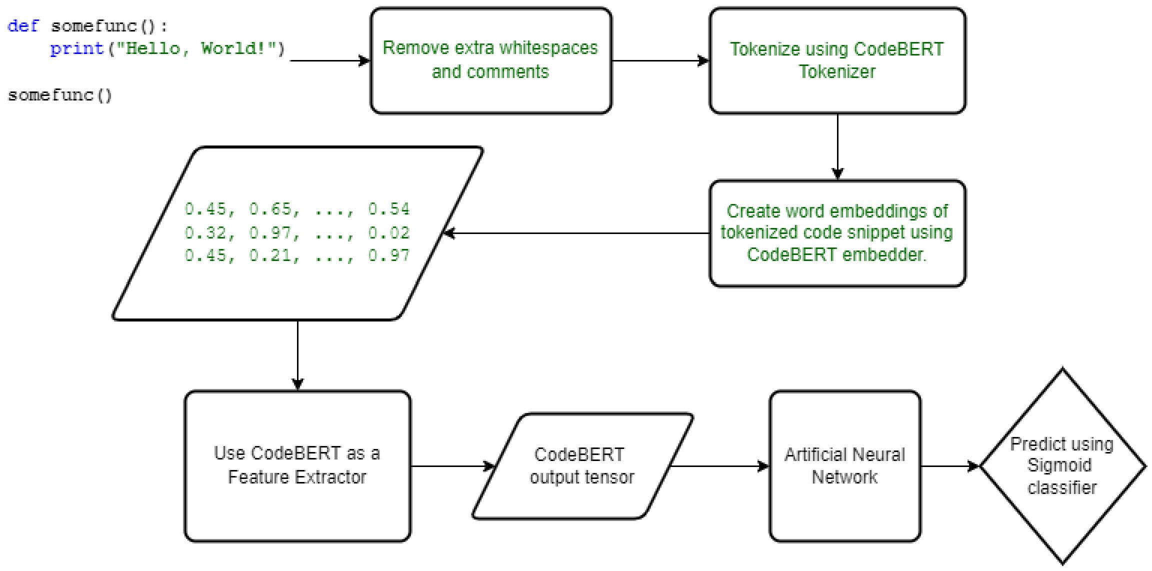
$$\kappa_0 = \frac{\bar{P} - P_e}{1 - P_e} + \frac{1 - \bar{P}}{Nm_0(1 - P_e)} \quad (4.5)$$

The extension of the kappa coefficient is an extension of Fleiss' Kappa Coefficient [17] by adding a proportion for multilabel classification wherein the total number of observations m_0 is taken into account.

4.2 Model, Training, and Evaluation

The researchers used PyTorch[43] library for implementation and PyTorch Lightning for training[13].

Furthermore, three loss functions were used during training, namely *Cross-entropy*, *Weighted Cross-entropy*, and *Focal Loss*[34] for better performance comparisons.


def somefunc():
    print("Hello, World!")

somefunc()
Remove extra whitespaces and comments
Tokenize using CodeBERT Tokenizer
Create word embeddings of tokenized code snippet using CodeBERT embedder.
0.45, 0.65, ..., 0.54
0.32, 0.97, ..., 0.02
0.45, 0.21, ..., 0.97
Use CodeBERT as a Feature Extractor
CodeBERT output tensor
Artificial Neural Network
Predict using Sigmoid classifier

Figure 4.3: Algorithm flowchart

4.2.1 General Flow of the Network

Input preprocessing

Before feeding into the neural network, a code snippet should be preprocessed into a format that would remove any extra whitespaces and comments. This is done through regular expressions.

When preprocessing Python code snippets, both the Python comment style which starts with ”`#`” and docstring-style comments (`"""This is a comment"""`) which begin and end with are removed.

In the same manner, Java and JavaScript comments can be removed with a single regex script since they share the same comment syntax.

Furthermore, extra whitespaces are to be removed. However, in the case of Python where whitespaces are essential for a functional and working program, it is considered not to remove these extra whitespaces.

Listing 4.1 shows a code snippet that has multiple whitespaces. However, since the CodeBERT model only accepts input length below 512, the input codes have to be made as small as possible. Note that the length pertains to the token, not to the character. Each whitespace will be considered as a single token and has to be minimized without changing the semantics of the program.

Listing 4.1: A code snippet with multiple whitespaces

```python
def this_has_many_whitespaces   (  whitespace)    :

    print( 'This is a    whitespace')
    # For some reason, this code has enough whitespaces
    #       in it.
```

Listing 4.2: Whitespaces removed

```python
def this_has_many_whitespaces(whitespace):
    print( 'This is a whitespace')
```

After removing the extra whitespaces and comments, the model make use of the existing CodeBERT tokenizer.[15] This maps every token to an integer which can be embedded into a word embedding which is also provided by CodeBERT. If a 1-D tokenized array is inputted into the embedding, the output will be a 2-D matrix of word embeddings. This is the input to CodeBERT

Feature Extraction

The word embeddings generated in the previous subsection are fed into CodeBERT and the latter is used to make sense of the data. This is known as feature extraction. In this study, CodeBERT is used as a feature extractor without further training CodeBERT on the dataset. The gradients on CodeBERT are frozen and the output of CodeBERT are become the inputs to a custom deep neural network, which will act as the classifier for a multilabel classification task. Freezing the gradients of CodeBERT and treating it as a feature extractor makes training faster and less expensive. The output of CodeBERT is a tensor of size $768 \times 512 \times batch\_size$, where the number 512 pertains to the input length, and 768 pertains to 768.

Artificial Neural Network

The model flattens the output of CodeBERT into a tensor of size $393,216 \times batch\_size$. Currently, this artificial neural network is very basic and does not have any hidden layers. From the input layer, the data is directly propagated into the output layer which has 9 neurons, one for each label. The neurons are activated by a sigmoid classifier since each label is mutually independent. However, since the sigmoid function suffers from the vanishing gradient problem, this can cause the model to have problems when backpropagating. This problem can be mitigated by adding batch normalization to limit the possible values in the activation neurons and prevent very low gradients.

4.2.2 Evaluation

Since most work on code summarization deals with single labels, they often use the BLEU score or the F1 metric[2]. However, because this paper deals with multi-label classification, a different metric, which evaluates multi-label problems more effectively, should be used. Hence, the researchers chose four metrics as stated in previous work on multi-label sentiment analysis[49].

For the following, let N be the set of data samples and L be the set of labels.

The first metric is the Hamming loss which is defined to be the fraction of wrong labels over the total labels.

$$HL = \frac{1}{NL} \sum_{i=1}^{N} \sum_{j=1}^{L} (y_{i,j} \oplus \widehat{y}_{i,j}) \tag{4.6}$$

Here, summing over the xor ($\oplus$) operation of the real label $y_{i,j}$ and the prediction $\widehat{y}_{i,j}$ only counts the mismatched labels.

The subset accuracy only considers the examples that have exact correct labels, and is therefore, the most strict metric.

Let $F(P, R) = \frac{2PR}{P+R}$ be the F-function. The two other metrics are micro-F1 and macro-F1. These metrics are defined using micro-recall, micro-precision, macro-recall, and macro-precision. If TP is the sum of all the true positives, TN is the sum of all true negatives, FP is the sum of false positives, and FN is the sum of all false negatives, for each label and prediction, then $micro_{recall} = \frac{TP}{TP+FN}$ and $micro_{precision} = \frac{TP}{TP+FP}$. The micro-F1 is defined as $micro - F1 = F(micro_{precision}, micro_{recall})$

Furthermore the recall is defined as $\frac{truepositive}{truepositive+falsenegative}$ while the precision is defined as $\frac{truepositive}{truepositive+falsepositive}$

To get the macro-recall and macro-precision, if L is the set of all labels, then the recall for label l is $recall_l$ and the precision is $precision_l$ using the above equations. Then, the macro-recall and macro-precisions are defined as follows: $macro_{recall} = \frac{\sum_{l \in L} recall_l}{|L|}$ and $macro_{precision} \frac{\sum_{l \in L} precisionl}{|L|}$

4.3 Research Questions

RQ1. *How effective would the CodeBERT Transformer model be in an algorithm classification task?*

The methodology used in 4.2.2 will be used to assess the performance of the CodeBERT. Each metric will be recorded while training and their final loss and accuracy will be plotted. The subset loss is a stringent loss function that punishes the model even when there is only one mislabel. The Hamming loss is a widely-used multilabel classifier metric since the subset loss may be too punishing for some applications[55].

RQ2. *Which algorithms does the Transformer perform best/worst on?*

Confusion matrices and bar plots should show which labels the model misclassify and classify properly. With this, the study can show which algorithms the model has trouble classifying properly. Furthermore, percentages of true positives will be measured for each label on the testing phase.

RQ3. *How would changing the loss function affect the performance of the algorithm detection model?*

The three loss functions: cross-entropy loss, weighted-cross entropy loss, and focal loss, will be used to evaluate the model at different times using the same methodology discussed above with k-fold validation.

Chapter 5

Results

5.1 Interrater Reliability

Extended kappa statistics results after labeling the data showed that the average proportion of observed agreement per code snippet is $\bar{P} = 0.847353$ while the proportion of agreement by chance under each category is $P_e = 0.107143$ and the total number of observations is $m_0 = 5206$. Since the average proportion of observed agreement per data point is high, and the proportion of expected agreement under each category is low, an almost perfect agreement of $\kappa_0 = 0.829035$ had been achieved among the labelers, this means that expert validation is no longer necessary 4.1.4.

Algorithm	p_j
Quicksort	0.121195
Mergesort	0.125254
Selectionsort	0.095923
Insertionsort	0.124331
Bubblesort	0.118982
Linear Search	0.107729
Binary Search	0.056078
Linked List	0.127467
Hashmap	0.083379

Table 5.1: Proportion of agreement per Category

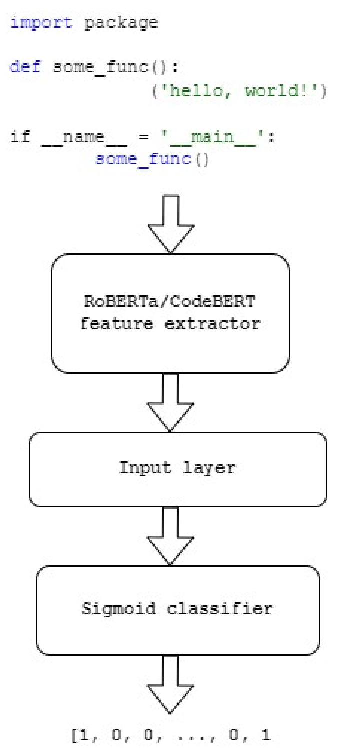
The proportion of agreement per category p_j is computed by the total number of data labeled under each category divided by the product of the total number of data in the dataset and the number of raters.

	Poor	Slight	Fair	Moderate	Substantial	Almost perfect
Kappa	0.0	0.0	0.40	0.60	0.80	1.0

κ coeff.	Agreement
≤ 0	Less than chance Agreement
0.01 to 0.20	Slight Agreement
0.21 to 0.40	Fair Agreement
0.41 to 0.60	Moderate Agreement
0.61 to 0.80	Substantial Agreement
0.81 to 0.99	Almost Perfect Agreement

Table 5.2: Interpretation of κ coeff.

5.2 Experimental Setup



```
import package

def some_func():
            ('hello, world!')

if __name__ = '__main__':
        some_func()
```

Figure 5.1: Model

The initial experiments are run on a Google Colab notebook running on a GPU backend. Unfortunately, due to the availability of GPUs for the Google Colab service, the GPU model may change every session.

Figure 5.1 shows how the model was set up. The input code snippet is tokenized and embedded using CodeBERT's embedder. This is then feed into CodeBERT which acts as a feature extractor. That is, its weights are removed and will no longer be trained. CodeBERT outputs a tensor of size $768 \times 512 \times batch\_size$ where 768 is the embedding size and 512 is the length of the input tokens. This output is then inputted into an artificial neural network with an input layer of $393,216$ neurons and an output layer with 9 neurons each activated by a sigmoid classifier with a threshold of 0.5.

The choice for model hyperparameters such as learning rate, number of layers, and number of neurons per layer are arbitrary and chosen by trial and error. The initial experiment uses the ADAM algorithm [28] with the default PyTorch parameters as the optimizing function using three different loss functions; namely, binary cross entropy, weighted binary cross entropy, and focal loss. The class weights are computed based on the least amount of data to the largest amount of data to compensate for the imbalance in the dataset.

The models were trained for seven epochs with a training batch size of 32 and tested with 8-Fold validation. The choice of $k = 8$ was done to assure that each model will have 12.5% validation set. Each fold, except for the last one, has a train set of 1610 examples and a validation set of 228 examples. The last fold has a train set of 1609 examples and a validation set of 229 examples.

Furthermore, the classifier component is an artificial neural network composed of two layers: the input layer with 420 neurons which takes in the output of CodeBERT and the output layer with 9 neurons activated by the sigmoid function which acts as the classifier.

The following table shows the entire training setup.

seed	1729
Learning rate	0.00001
Data type	float32
Training batch size	32
Validation batch size	2
Logging every n steps	8
Number of splits in K-Fold	8

Table 5.3: Training Setup

5.3 Training Results

To empirically evaluate the model, 24 different models were trained and evaluated on the same dataset. The researchers made use of three loss functions to compare the model performance on the K-Fold validation set. Each fold was trained for 30 minutes. The time taken for computing the validation set is 10 minutes. With each setup time having an average of 3 minutes, the total training time for all 8-folds for each loss function is 5.4 hours.

Figure 5.2 shows the averaged training losses of all 8 folds for each loss function. It is shown that focal loss had lower training loss compared to cross entropy and weighted cross entropy losses. This can be mostly attributed to the former giving less weight to the easier examples to label, as stated in the original paper. However, it is shown in the plot that all three models converge.

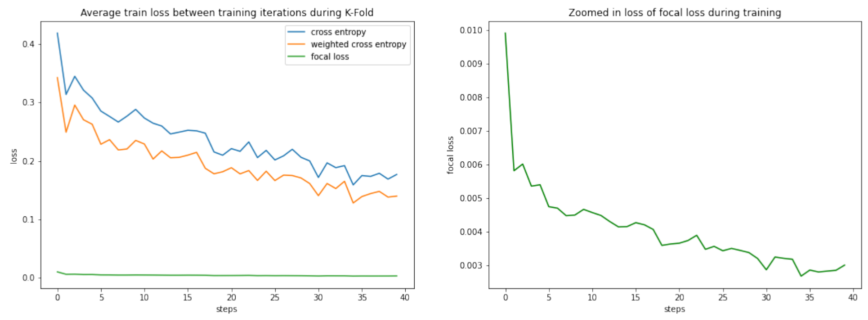


Figure 5.2: Average train loss between training iterations

The following plots show the different training loss along with the validation losses.

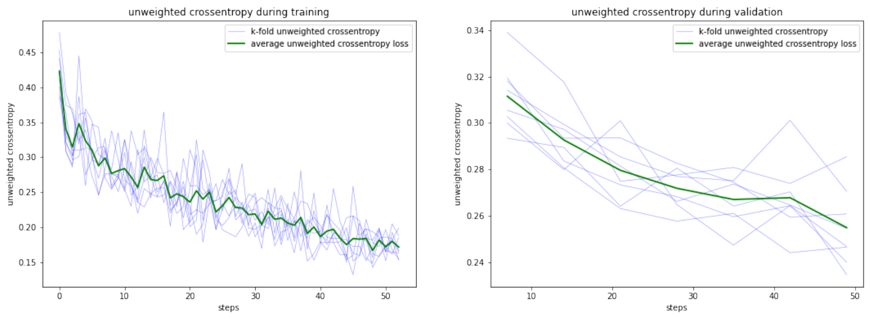


Figure 5.3: Average cross-entropy Loss during training and validation

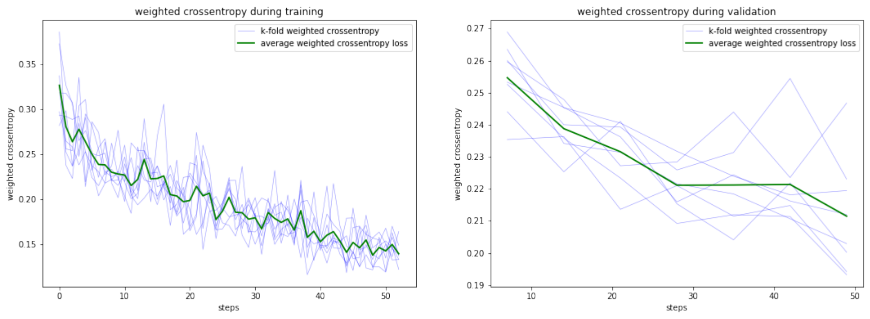


Figure 5.4: Average weighted cross-entropy loss during training and validation

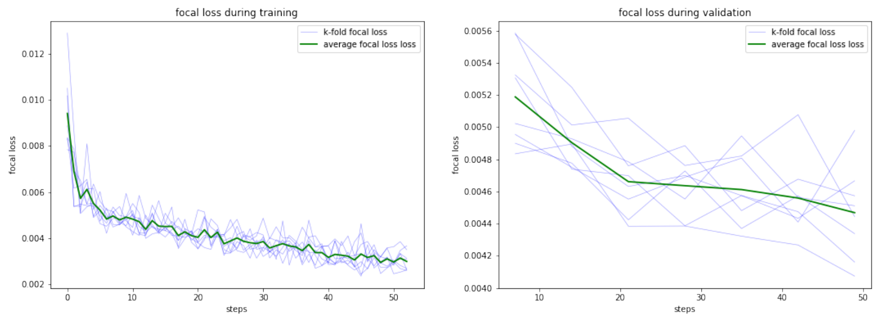


Figure 5.5: Average focal loss during training and validation

Both the binary cross entropy loss and its weighted version show a volatile training loss, however, decreasing, which could be the effect of the learning rate. The training loss of focal loss shows less volatility, however. All three loss functions have an almost constant validation loss which is a sign of underfitting. Subsequent experiments would aim to overcome this problem by adding a hidden layer and by adding more epochs during training 5.4. A detailed account of all three loss functions is stated in the appendix.

		Cross Entropy Loss models	Weighted Cross Entropy Loss models	Focal Loss models
Hamming loss	avg	0.09566	0.09706	0.11145
	std	0.00453	0.00477	0.01014
Subset acc	avg	0.25481	0.2482	0.27273
	std	0.02652	0.03533	0.03584
Micro F1	avg	0.26724	0.27374	0.41683
	std	0.05357	0.06033	0.04237
Macro F1	avg	0.24734	0.24274	0.37846
	std	0.04815	0.05653	0.04515
Loss function	avg	0.25484	0.21147	0.00447
	std	0.01685	0.01802	0.00029

Table 5.4: Validation Metrics

Table 5.4 records the average and standard deviation of validation metrics after the models were fully trained and evaluated on the same validation set. The results of each fold can be found in the appendix C.

1. **Hamming Loss** is the ratio of the number of wrong predictions over total predictions, this is used to estimate how often the model is making errors when predicting. According to the results, the models trained using weighted-cross entropy loss had the lowest hamming loss, and therefore, made the least mistakes when predicting. Furthermore, focal loss has the most mistakes.

2. **Subset Accuracy** is the strictest of all the metrics above. That is, either all predictions are correct or they are wrong. All three models show an average subset accuracy of 24%-27%. However, since a significant amount of data have all 0s, this metric is not as significant as the F1 metrics.

3. **F1 Score.** The model trained with focal loss had a higher micro-averaging but lower macro-averaging. Otherwise, all models show a lower micro-F1 score due to the fact that each label is treated equally, whereas, macro-F1 score treats each label independently.

Furthermore, the metrics in Table 5.5 show the scores over the entire dataset. These scores were computed post-training by aggregating all the predictions from the validation sets during the 8-fold validation, whereas, the F1 scores from Table 5.4 were the averaged F1 scores across all 8-fold validation sets.

	Cross Entropy Loss models	Weighted Cross Entropy Loss models	Focal Loss models
Micro-Precision	0.8006	0.77183	0.52097
Micro-Recall	0.16483	0.16667	0.34252
Micro-F1	0.27337	0.27414	0.41331
Macro-Precision	0.61447	0.63137	0.52106
Macro-Recall	0.16103	0.15521	0.32826
Macro-F1	0.25518	0.24917	0.40278

Table 5.5: Validation metrics over the entire dataset

When a student submits a code submission in CodeC's constraint checker, the latter only checks the constraints that are in the submission. That is, if the model wrongly predicts a false positive of an algorithm that is not included in the constraints, the constraint checker will not care about that. With this, the recall is an important metric for this study.

Table 5.4 shows that Cross Entropy Loss models and Weighted Cross Entropy Loss models show a high precision but low recall. The researchers attribute this due to the highly significant amount of *FALSE* labels which biases the model towards *FALSE*. In contrast, focal loss models' precision is lower. That is, it is predicting false positives. But it has a significant recall advantage over the other models due to giving the *TRUE* labels more weight when setting the focal loss hyper-parameter $\gamma = 5$. However, the recall is still lower than what the researchers intended. These results are reflected in the following plots.

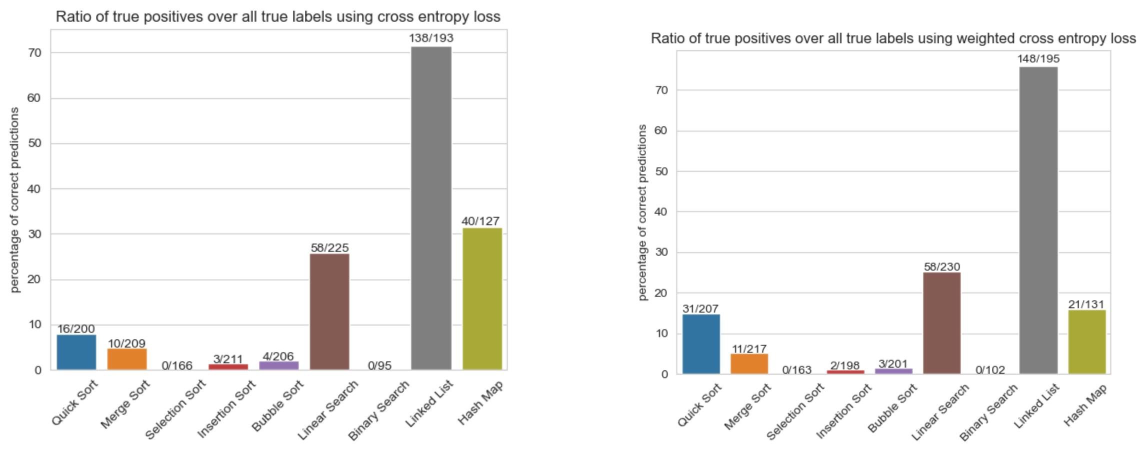


Figure 5.6: Ratio of true positives over all true labels using Cross Entropy and Weighted Cross Entropy Loss

In the Figure 5.6, most of the labels seem to be very hard to classify except for Linked List which has more than 70% true positive rate. On the other hand, the model trained on focal loss showed more favorable results.

5.4 Future Experiments

The researchers aim to overcome underfitting and increasing recall on all three models by introducing hidden layers, extending training time, and a learning rate scheduler. Also, decreasing the threshold of the sigmoid classifier may be done to improve recall.

In addition, an additional and smaller dataset will be collected to measure how the model

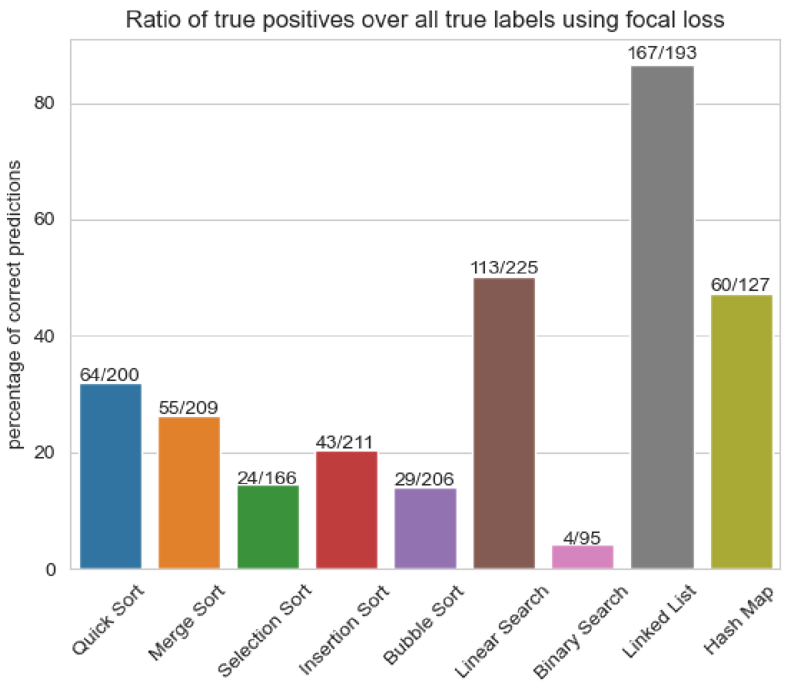


Figure 5.7: Ratio of true positives over all true labels using Focal Loss

performs given the foreground (the part of the code that will need to be classified by the classifier) and the background data (the part of the code that is not part of any of the algorithms that need to be classified). This dataset will not be part of the training set or validation set.

Bibliography

[1] Uri Alon et al. "code2seq: Generating sequences from structured representations of code". In: *arXiv preprint arXiv:1808.01400* (2018).

[2] Uri Alon et al. "code2vec: Learning distributed representations of code". In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–29.

[3] Uri Alon et al. "Structural language models of code". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 245–256.

[4] Agathe Balayn and Alessandro Bozzon. "Designing evaluations of machine learning models for subjective inference: The case of sentence toxicity". In: *arXiv preprint arXiv:1911.02471* (2019).

[5] Long Chen, Wei Ye, and Shikun Zhang. "Capturing source code semantics via tree-based convolution over API-enhanced AST". In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 2019, pp. 174–182.

[6] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[7] *Codechef Competitive Programming — Kaggle*. URL: `https://www.kaggle.com/arjoonn/codechef-competitive-programming` (visited on 11/21/2021).

[8] Daniel Coore and Daniel Fokum. "Facilitating course assessment with a competitive programming platform". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019, pp. 449–455.

[9] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[10] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[11] William Di Luigi et al. "Learning analytics in competitive programming training systems". In: *2018 22nd International Conference Information Visualisation (IV)*. IEEE. 2018, pp. 321–325.

[12] Tania Di Mascio, Luigi Laura, and Marco Temperini. "A framework for personalized competitive programming training". In: *2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET)*. IEEE. 2018, pp. 1–8.

[13] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Mar. 2019. DOI: `10.5281/zenodo.3828935`. URL: `https://github.com/PyTorchLightning/pytorch-lightning`.

[14] Steven Y Feng et al. "A survey of data augmentation approaches for nlp". In: *arXiv preprint arXiv:2105.03075* (2021).

[15] Zhangyin Feng et al. "Codebert: A pre-trained model for programming and natural languages". In: *arXiv preprint arXiv:2002.08155* (2020).

[16] Gerhard Fischer. "Distributed intelligence: extending the power of the unaided, individual human mind". In: *Proceedings of the working conference on Advanced visual interfaces*. 2006, pp. 7–14.

[17] Joseph L Fleiss. "Measuring nominal scale agreement among many raters." In: *Psychological bulletin* 76.5 (1971), p. 378.

[18] Olly Gotel, Christelle Scharff, and Andrew Wildenberg. "Teaching software quality assurance by encouraging student contributions to an open source web-based system for the assessment of programming assignments". In: *Proceedings of the 13th annual conference on Innovation and technology in computer science education*. 2008, pp. 214–218.

[19] Olly Gotel, Christelle Scharff, and Andy Wildenberg. "Extending and contributing to an open source web-based system for the assessment of programming problems". In: *Proceedings of the 5th international symposium on Principles and practice of programming in Java*. 2007, pp. 3–12.

[20] Olly Gotel et al. "Global perceptions on the use of WeBWorK as an online tutor for computer science". In: *2008 38th Annual Frontiers in Education Conference*. IEEE. 2008, T4B–5.

[21] Sugandha Gupta and Anamika Gupta. "E-Assessment Tools for Programming Languages: A Review." In: *ICITKM*. 2017, pp. 65–70.

[22] Georgiana Haldeman et al. "CSF: Formative Feedback in Autograding". In: *ACM Transactions on Computing Education (TOCE)* 21.3 (2021), pp. 1–30.

[23] Geoffrey E Hinton et al. "Learning distributed representations of concepts". In: *Proceedings of the eighth annual conference of the cognitive science society*. Vol. 1. Amherst, MA. 1986, p. 12.

[24] Charles AR Hoare. "Quicksort". In: *The Computer Journal* 5.1 (1962), pp. 10–16.

[25] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[26] Hamel Husain et al. *CodeSearchNet Challenge: Evaluating the State of Semantic Code Search*. 2020. arXiv: `1909.09436 [cs.LG]`.

[27] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. "Towards a systematic review of automated feedback generation for programming exercises". In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 2016, pp. 41–46.

[28] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[29] Helena C Kraemer. "Kappa coefficient". In: *Wiley StatsRef: statistics reference online* (2014), pp. 1–4.

[30] Helena Chmura Kraemer. "Extension of the kappa coefficient". In: *Biometrics* (1980), pp. 207–216.

[31] H Chad Lane and Kurt VanLehn. "Teaching the tacit knowledge of programming to noviceswith natural language tutoring". In: *Computer Science Education* 15.3 (2005), pp. 183–201.

[32] Rael T Lange. "Inter-rater reliability". In: *Encyclopedia of Clinical Neuropsychology. New York, NY: Springer New York* (2011), p. 1348.

[33] Nguyen-Thinh Le, Wolfgang Menzel, and Niels Pinkwart. "Evaluation of a constraint-based homework assistance system for logic programming". In: *Proceedings of the 17th International Conference on Computers in Education.* 2009.

[34] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision.* 2017, pp. 2980–2988.

[35] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[36] Yihan Lu and I-Han Hsiao. "Modeling semantics between programming codes and annotations". In: *Proceedings of the 29th on Hypertext and Social Media.* 2018, pp. 101–105.

[37] Anshuman Majumdar. "MacACM: Encouraging competitive programming via mentorship and outreach". In: *XRDS: Crossroads, The ACM Magazine for Students* 24.1 (2017), pp. 14–15.

[38] Mary L McHugh. "Interrater reliability: the kappa statistic". In: *Biochemia medica* 22.3 (2012), pp. 276–282.

[39] Lili Mou et al. "Convolutional neural networks over tree structures for programming language processing". In: *Thirtieth AAAI Conference on Artificial Intelligence.* 2016.

[40] Prashant R Nair. "Increasing Employability of Indian Engineering Graduates through Experiential Learning Programs and Competitive Programming: Case Study". In: *Procedia Computer Science* 172 (2020), pp. 831–837.

[41] Ágnes ERDőSNé NéMETH and László ZSAKÓ. *Grading Systems for Algorithmic Contests.* 2018.

[42] Martina Öqvist and Jalal Nouri. "Coding by hand or on the computer? Evaluating the effect of assessment mode on performance of students learning programming". In: *Journal of Computers in Education* 5.2 (2018), pp. 199–219.

[43] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019), pp. 8026–8037.

[44] Chris Piech et al. "Learning program embeddings to propagate feedback on student code". In: *International conference on machine Learning.* PMLR. 2015, pp. 1093–1102.

[45] Albert Pritzkau. “NLytics at CheckThat! 2021: multi-class fake news detection of news articles and domain identification with RoBERTa-a baseline model”. In: *Faggioli et al.[33]* (2021).

[46] Robert Sedgewick and Kevin Wayne. *Algorithms.* Addison-wesley professional, 2011.

[47] Steven S Skiena. *The algorithm design manual.* Springer International Publishing, 2020.

[48] Rifa Zakia Tanvir Ahmed. “Exploring the Formats of Programming Context”. In: ().

[49] Jie Tao and Xing Fang. “Toward multi-label sentiment analysis: a transfer learning based approach”. In: *Journal of Big Data* 7.1 (2020), pp. 1–26.

[50] Juan Pablo Usuga-Cadavid et al. “Exploring the Influence of Focal Loss on Transformer Models for Imbalanced Maintenance Data in Industry 4.0”. In: *IFAC-PapersOnLine* 54.1 (2021), pp. 1023–1028.

[51] Ashish Vaswani et al. “Attention is all you need. CoRR abs/1706.03762 (2017)”. In: *arXiv preprint arXiv:1706.03762* (2017).

[52] Petar Veličković et al. “The CLRS Algorithmic Reasoning Benchmark”. In: (2021).

[53] Yutaka Watanobe et al. “Next-Generation Programming Learning Platform: Architecture and Challenges”. In: *SHS Web of Conferences.* Vol. 77. EDP Sciences. 2020, p. 01004.

[54] Huihui Wei and Ming Li. “Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code.” In: *IJCAI.* 2017, pp. 3034–3040.

[55] Guoqiang Wu and Jun Zhu. “Multi-label classification: do Hamming loss and subset accuracy really conflict with each other?” In: *arXiv preprint arXiv:2011.07805* (2020).

[56] Rui Xie et al. “Exploiting Method Names to Improve Code Summarization: A Deliberation Multi-Task Learning Approach”. In: *arXiv preprint arXiv:2103.11448* (2021).

[57] Yuto Yoshizawa and Yutaka Watanobe. “Logic error detection system based on structure pattern and error degree”. In: *Advances in Science, Technology and Engineering Systems Journal* 4.5 (2019), pp. 1–15.

[58] Hana Yousuf et al. “A systematic review on sequence-to-sequence learning with neural network and its models.” In: *International Journal of Electrical & Computer Engineering (2088-8708)* 11.3 (2021).

Appendix A

Algorithms

A.1 Target Algorithms and Guidelines

This section deals with how each target label will be labeled based on pseudocode and functionality on a technical level. Note that the pseudocodes in this section is non-exhaustive as different programmers may choose different ways to implement their code. Pseudocode sources will include algorithms textbooks such as Introduction to Algorithms[9], Algorithm Design Manual [47], and Algorithms[46]; and original papers defining the algorithm by the original authors. These primary and secondary resources should be consulted whenever there is a doubt.

A.1.1 Linear Search

Linear search algorithms that will be considered would be using for loops, for each loops, and while loops as implemented below,

Algorithm 1 $linear - search - for(A, k)$

```
Input: $A$: array, $k$: key
  for $i = 1$ to $A.length$ do
    if $A[i] = k$ then
      return $i$
    end if
  end for
  return $-1$
```

Algorithm 2 $linear - search - foreach(A, k)$

```
Input: $A$: array, $k$: key
  $j \leftarrow 1$
  for $item$ in $A$ do
    if $item = k$ then
      return $j$
    else
      $j \leftarrow j + 1$
    end if
  end for
  return $-1$
```

Algorithm 3 $linear - search - while(A, k)$

```
Input: $A$: array, $k$: key
  $j \leftarrow 1$
  while $j \leq A.length$ do
    if $A[j] = k$ then
      return $j$
    end if
    $j \leftarrow j + 1$
  end while
  return $-1$
```

These implementations may be evident in searching through arrays, vectors, matrices, or linked lists.

A.1.2 Binary Search

From the Algorithm Design Manual [47] a binary search algorithm is one that generally follows the following code (note that variations may occur):

```
Algorithm 4 binary_search($A, k, p, q$)
Input: $A$: array, $k$: key, $p$: index, $q$: index
Output: $A$ is sorted
  if $p > q$ then
      return $-1$
  end if
  $m \leftarrow \frac{p+q}{2}$
  if $A[m] = k$ then
      return $m$
  else if $A[m] > k$ then
      return binary_search($A, k, p, m-1$)
  else
      return binary_search($A, k, m+1, q$)
  end if
```

It is also possible that the binary search may not be recursive. In that case, it should be in the general form of

Algorithm 5 *binary_search*(A, k)

```
Input: $A$: Array, $k$: key
Output: $A$ is sorted
  Let $begin \leftarrow 0$ and $mid \leftarrow 0$
  Let $last \leftarrow |A| - 1$
  while $begin \leq last$ do
      $mid \leftarrow \frac{begin + last}{2}$
      if $A[mid] < k$ then
          $begin \leftarrow mid + 1$
      else if $A[mid] > k$ then
          $last \leftarrow mid - 1$
      else
          return $mid$
      end if
  end while
  return $-1$
```

If the given code contains either of the binary search general forms, it should be marked as having implemented binary search.

A.1.3 Insertion Sort

The textbook Introduction to Algorithms implement *insertion sort* as follows

Algorithm 6 $insertion - sort(A)$

```
Input: $A$: array
  for $j = 2$ to $A.length$ do
    $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$ do
      $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    end while
    $A[i+1] \leftarrow key$
  end for
```

A.1.4 Bubble Sort

The textbook Introduction to Algorithms implement *bubble sort* as follows

Algorithm 7 $bubble - sort(A)$

```
Input: $A$: array
  for $i = 1$ to $A.length - 1$ do
    for $j = A.length$ decrement to $i + 1$ do
      if $A[j] < A[j-1]$ then
        swap $A[j]$ with $A[j-1]$
      end if
    end for
  end for
```

A.1.5 Quick Sort

The quicksort algorithm can be implemented using either the Hoare partition or the Lomuto partition. The recursive code snippet below can use both partition functions.

Algorithm 8 *quicksort*(A, p, r)

```
Input: $A$: An array, $p$: index, $r$: index
  if $p < r$ then
      $q \leftarrow partition(A, p, r)$
      $quicksort(A, p, q - 1)$
      $quicksort(A, q + 1, r)$
  end if
```

In the original quicksort paper[24], the partition function, called the Hoare-partition, is implemented as follows:

Algorithm 9 *partition*(A, p, r)

```
Input: $A$: An array, $p$: index, $r$: index
  $x \leftarrow A[p]$
  $i \leftarrow p - 1$
  $j \leftarrow r + 1$
  while TRUE do
      repeat
          $j \leftarrow j - 1$
      until $A[j] \leq x$
      repeat
          $i \leftarrow i + 1$
      until $A[i] \geq x$
      if $i < j$ then
          swap $A[i]$ and $A[j]$
      end if
  end while
```

However, another implementation of the partition function can be implemented as follows[9]

Algorithm 10 *partition*(A, p, r)

```
Input: $A$: An array, $p$: index, $r$: index
  $x \leftarrow A[r]$
  $i \leftarrow p - 1$
  for $j = p$ to $r - 1$ do
      if $A[j] \leq x$ then
          $i \leftarrow i + 1$
          swap $A[i]$ and $A[j]$
      end if
  end for
  swap $A[i + 1]$ and $A[r]$
  return i + 1
```

A.1.6 Merge Sort

The textbook CLRS implements the recursive mergesort by first defining a merge function and then the recursive mergesort function.

Algorithm 11 $merge(A, p, q, r)$

```
Input: $A$: An array, $p$: index, $q$: index, $r$: index
  $n_1 \leftarrow q - p + 1$
  $n_2 \leftarrow r - 1$
  $L[1 : n_1 + 1] \leftarrow A[p : p + n_1 - 1]$
  $R[1 : n_2 + 1] \leftarrow A[q + 1 : q + n_2]$
  $L[n_1 + 1], R[n_2 + 1] \leftarrow \infty, \infty$
  $i, j \leftarrow 1, 1$
  for $k = p$ to $r$ do
    if $L[i] \leq R[j]$ then
      $A[k] \leftarrow A[i]$
      $i \leftarrow i + 1$
    else
      $A[k] \leftarrow R[j]$
      $j \leftarrow j + 1$
    end if
  end for
```

It is now possible to use the `merge` operation inside a recursive mergesort.

Algorithm 12 $merge - sort(A, p, r)$

```
Input: $A$: An array, $p$: index, $r$: index
  if $p < r$ then
    $q \leftarrow \lfloor (p + r)/2 \rfloor$
    $merge - sort(A, p, q)$
    $merge - sort(A, q + 1, r)$
    $merge(A, p, q, r)$
  end if
```

A.1.7 Selection Sort

In the Algorithm Design Manual,[47] the selection sort algorithm is defined as

Algorithm 13 *selection − sort*(A)

```
Input: $A$: An array
  for $i = 1$ to $n$ do
      $min \leftarrow i$
      for $j = i + 1$ to $n$ do
          if $A[j] < A[min]$ then
              $min \leftarrow j$
          end if
      end for
      if $min \neq i$ then
          swap $A[min]$ and $A[i]$
      end if
  end for
```

A.1.8 Linked List

The textbook Introduction to Algorithms[9] classifies *Linked Lists* as dictionary data structures such that it implements some operations that query or modifies the data structure. In this research, we restrict the operations three operations: `search`, `insert`, and `delete`. If a Linked List implementation implements these three operations, then we consider the code snippet as a linked list. Additional operations such as `length`, `minimum`, and `successor` will not be evaluated and are optional.

Linked Lists are usually implemented in imperative programming languages as structs or classes. There should be a `NODE` struct and a `LinkedList` struct. The `NODE` has the properties `KEY` and `NEXT`. The property `PREV` is optional. the `LinkedList` has the property `HEAD`. The properties `TAIL` and `LENGTH` are optional.

The Linked list search usually implements a linear search algorithm as follows:

Algorithm 14 $list - search(l, k)$

```
Input: $l$: The list, $k$: key to be searched
  $x \leftarrow l.head$
  while $x \neq NIL$ and $x.key \neq k$ do
      $x \leftarrow x.next$
  end while
  return $x$
```

Next is the implementation of an insert operation on a singly-linked list with a tail.

Algorithm 15 $list - insert(l, n)$

```
Input: $l$: The list, $n$: node to be inserted
  if $l.head = NIL$ then
      $l.head \leftarrow n$
      $l.tail \leftarrow n$
  else
      $l.tail.next \leftarrow n$
      $l.tail = n$
  end if
```

The delete of a Linked List often implements the following code snippet:

Algorithm 16 $list - delete(l, n)$

```
Input: $l$: The list, $n$: pointer to the node to be deleted.
  $x \leftarrow l.head$
  while $x \neq n$ do
      if $x.next = n$ then
          $x.next \leftarrow n.next$
      end if
  end while
```

Note that variations of these sample pseudocodes can vary. Double-linked lists and circular linked lists implemented with sentinels should also be accepted with linked list. Likewise, the linked list search operation should be accepted as a linear search operation since it displays the behavior of a linear search. Consult Introduction to Algorithms [9] for more comprehensive information on whether a code snippet implements a linked list.

A.1.9 Hashmap

Like the Linked List, Hashmap implementations should contain a `search`, `insert`, and `delete` operations. However, due to the different ways to implement hashmaps, this document ommits these technicalities. Hashmap data structures can be considered whenever they use any hashing functions during the three operations listed above. This hashing function could be implemented for any data type such as strings, ints, etc.

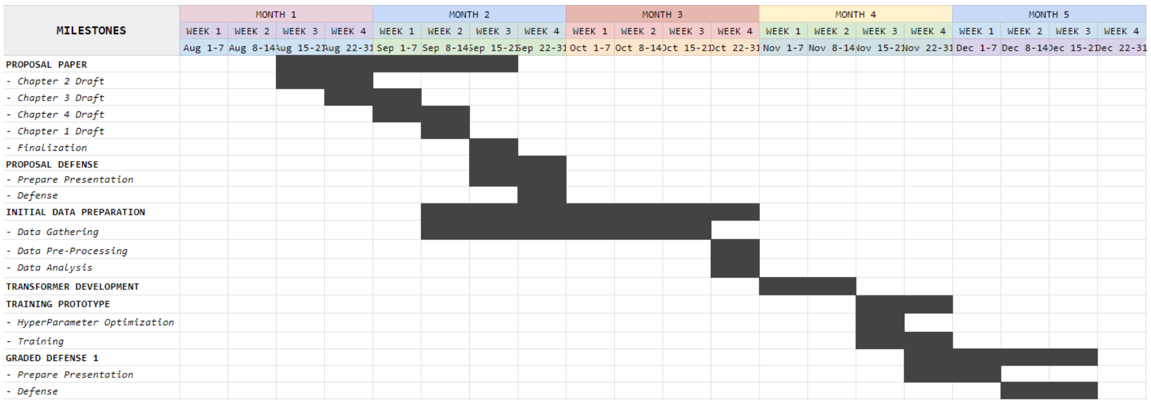
For collision avoidance, different methods can be considered as a hashmap such as using linked lists, binary trees, open addressing, or perfect hashing. For more information, refer to Introduction to Algorithms for introductory hashmap implementations.[9]

A.1.10 Otherwise case

The researchers added random code snippets such as the Euclidean algorithm and other algorithms to ensure that there are cases where the model does not pick any of the target labels. When this happens, everything should be marked zero.

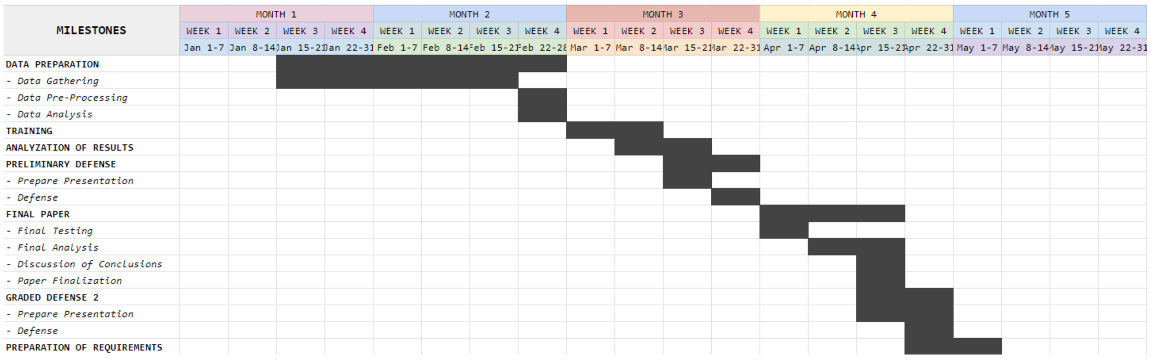
Appendix B

Research Timeline



MILESTONES	MONTH 1				MONTH 2				MONTH 3				MONTH 4				MONTH 5			
	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4
	Aug 1-7	Aug 8-14	Aug 15-21	Aug 22-31	Sep 1-7	Sep 8-14	Sep 15-21	Sep 22-31	Oct 1-7	Oct 8-14	Oct 15-21	Oct 22-31	Nov 1-7	Nov 8-14	Nov 15-21	Nov 22-31	Dec 1-7	Dec 8-14	Dec 15-21	Dec 22-31
PROPOSAL PAPER			■	■	■	■	■													
- Chapter 2 Draft			■	■																
- Chapter 3 Draft				■	■															
- Chapter 4 Draft					■	■														
- Chapter 1 Draft						■														
- Finalization							■													
PROPOSAL DEFENSE							■	■												
- Prepare Presentation							■	■												
- Defense								■												
INITIAL DATA PREPARATION						■	■	■	■	■	■	■								
- Data Gathering						■	■	■	■	■	■									
- Data Pre-Processing												■								
- Data Analysis												■								
TRANSFORMER DEVELOPMENT													■	■						
TRAINING PROTOTYPE															■	■				
- HyperParameter Optimization															■					
- Training															■	■				
GRADED DEFENSE 1																■	■	■	■	
- Prepare Presentation																■	■			
- Defense																		■	■	

First Semester Timeline



MILESTONES	MONTH 1				MONTH 2				MONTH 3				MONTH 4				MONTH 5			
	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 1	WEEK 2	WEEK 3	WEEK 4
	Jan 1-7	Jan 8-14	Jan 15-21	Jan 22-31	Feb 1-7	Feb 8-14	Feb 15-21	Feb 22-28	Mar 1-7	Mar 8-14	Mar 15-21	Mar 22-31	Apr 1-7	Apr 8-14	Apr 15-21	Apr 22-31	May 1-7	May 8-14	May 15-21	May 22-31
DATA PREPARATION			■	■	■	■	■	■												
- Data Gathering			■	■	■	■	■													
- Data Pre-Processing								■												
- Data Analysis								■												
TRAINING									■	■										
ANALYZATION OF RESULTS										■	■									
PRELIMINARY DEFENSE											■	■								
- Prepare Presentation											■									
- Defense												■								
FINAL PAPER													■	■	■					
- Final Testing													■							
- Final Analysis														■	■					
- Discussion of Conclusions															■					
- Paper Finalization															■					
GRADED DEFENSE 2															■	■				
- Prepare Presentation															■	■				
- Defense																■				
PREPARATION OF REQUIREMENTS																■	■			

Second Semester Timeline

Appendix C

Confusion Matrices

In this chapter, each confusion matrix has 4 elements.

- The *Upper Left* contains the **True Negative**.
- The *Upper Right* contains the **False Positive**.
- The *Lower Left* contains the **False Negative**.
- The *Lower Right* contains the **True Positive**.

The numbers in the four elements correspond to how many predictions of that sort were made by the model. To create the confusion matrix, the validation sets of all folds during the k-fold cross validation were collected and aggregated into a single and larger matrix.

C.1 Cross Entropy Loss

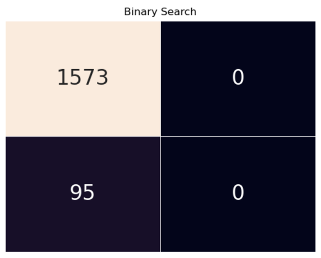


Figure C.1: CEL Binary Search

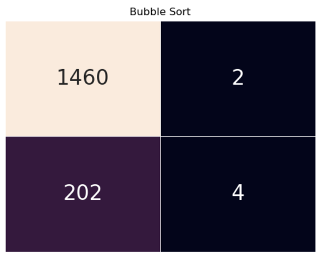


Figure C.2: CEL Bubble Sort

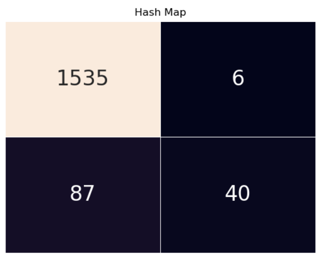


Figure C.3: CEL Hash Map

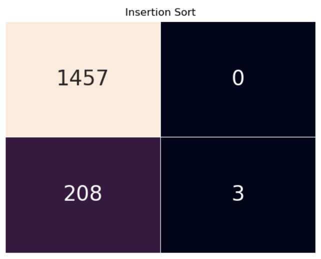


Figure C.4: CEL Insertion Sort

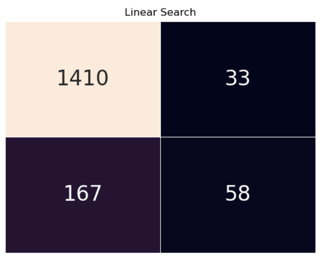


Figure C.5: CEL Linear Search

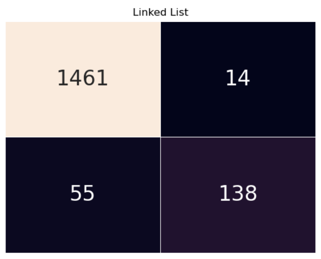


Figure C.6: CEL Linked List

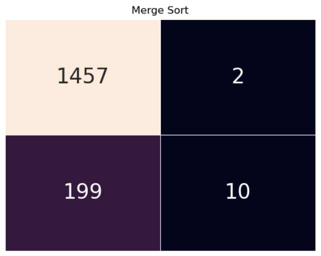


Figure C.7: CEL Merge Sort

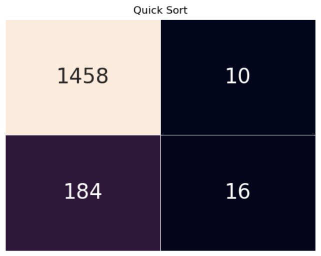


Figure C.8: CEL Quick Sort

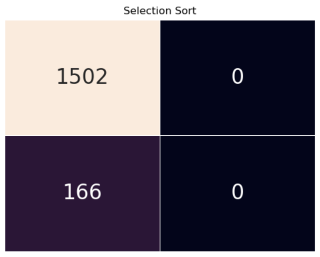


Figure C.9: CEL Selection Sort

C.2 Weighted Cross Entropy Loss

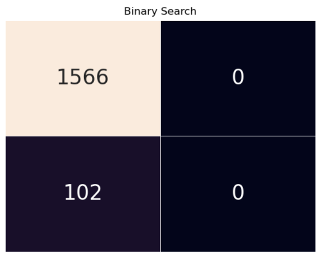


Figure C.10: WCEL Binary Search

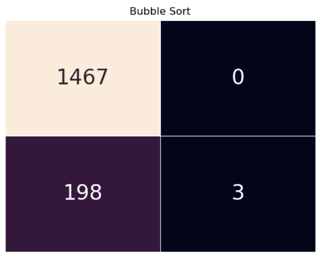


Figure C.11: WCEL Bubble Sort

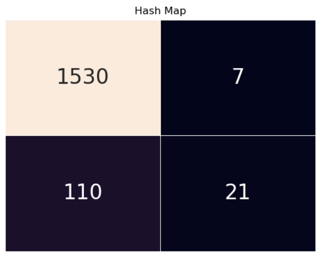


Figure C.12: WCEL Hash Map

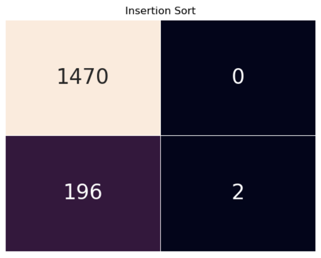


Figure C.13: WCEL Insertion Sort

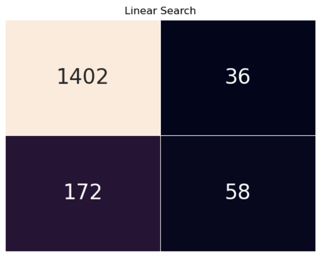


Figure C.14: WCEL Linear Search

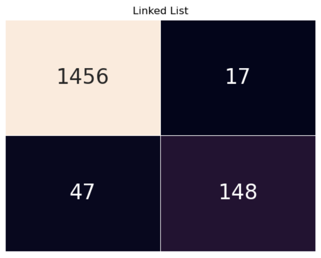


Figure C.15: WCEL Linked List

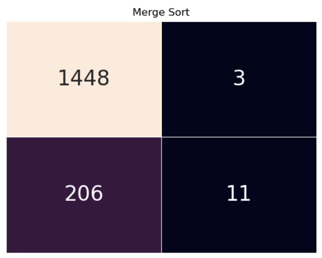


Figure C.16: WCEL Merge Sort

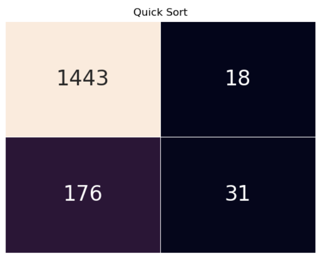


Figure C.17: WCEL Quick Sort

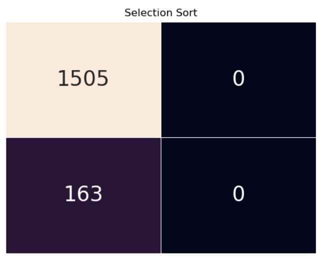


Figure C.18: WCEL Selection Sort

C.3 Focal Loss

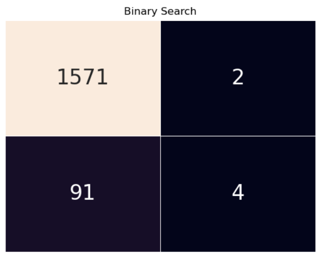


Figure C.19: FL Binary Search

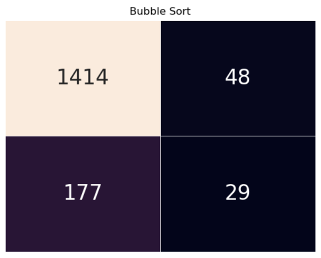


Figure C.20: FL Bubble Sort

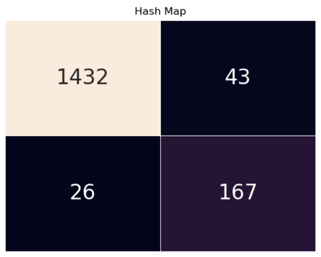


Figure C.21: FL Hash Map

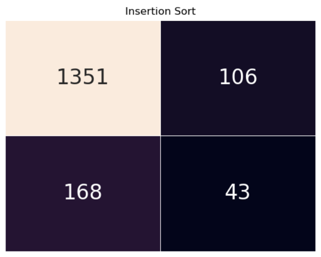


Figure C.22: FL Insertion Sort

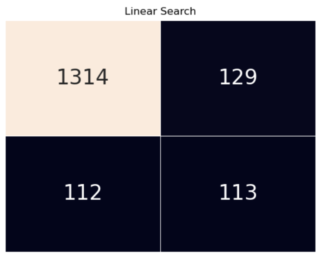


Figure C.23: FL Merge Sort

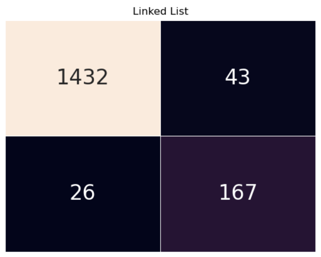


Figure C.24: FL Quick Sort

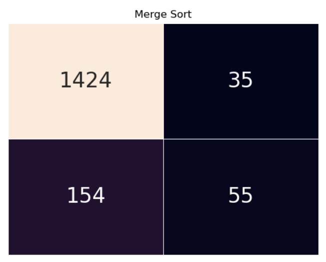


Figure C.25: FL Merge Sort

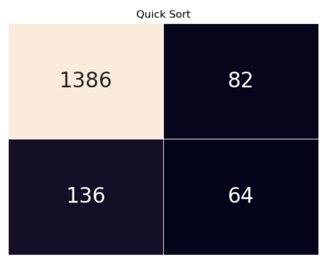


Figure C.26: FL Quick Sort

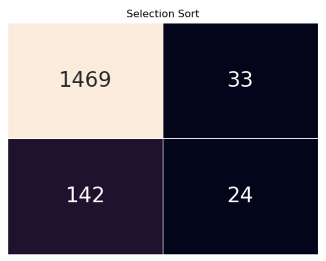


Figure C.27: FL Selection Sort

Appendix D

Loss Results

This chapter records the averages and standard deviation of the evaluation metrics for each epoch and each fold.

D.1 Cross-entropy Loss

epoch	1	2	3	4	5	6	7
fold 1	0.1095	0.1073	0.1015	0.1020	0.0972	0.1047	0.0935
fold 2	0.1031	0.1073	0.1015	0.1052	0.1057	0.1073	0.1031
fold 3	0.1052	0.0972	0.0972	0.09622	0.0951	0.09303	0.0898
fold 4	0.1031	0.1031	0.1026	0.09994	0.0962	0.0903	0.0967
fold 5	0.1169	0.1169	0.1025	0.1116	0.1020	0.1014	0.0998
fold 6	0.1132	0.0998	0.10149	0.10523	0.10309	0.09668	0.09134
fold 7	0.1009	0.096688	0.097756	0.09561	0.094017	0.090277	0.09134
fold 8	0.09989	0.10149	0.10416	0.09935	0.09401	0.10470	0.09294
avg	0.10651	0.10378	0.10112	0.101920	0.09845	0.098587	0.095656
std	0.00613	0.00669	0.00239	0.00533	0.00454	0.00688	0.00452

Table D.1: Cross-Entropy Model Hamming Loss

epoch	1	2	3	4	5	6	7
fold 1	0.06007	0.1851	0.1719	0.1877	0.3194	0.2189	0.2831
fold 2	0.1417	0.0822	0.1631	0.1916	0.2216	0.2021	0.2277
fold 3	0.0577	0.1655	0.1852	0.2225	0.1866	0.2434	0.2279
fold 4	0.0780	0.1083	0.0815	0.1228	0.1525	0.2644	0.1712
fold 5	0.0341	0.1719	0.2065	0.2155	0.2860	0.2621	0.2401
fold 6	0.0427	0.2284	0.2098	0.1436	0.2497	0.2565	0.3367
fold 7	0.0989	0.1846	0.1986	0.2137	0.2448	0.2660	0.2590
fold 8	0.1076	0.1045	0.1312	0.1592	0.1757	0.1601	0.2326
avg	0.0776	0.1538	0.1685	0.1821	0.2295	0.2342	0.2473
std	0.03639	0.0500	0.04364	0.0366	0.0568	0.0378	0.0481

Table D.2: Cross-Entropy Model Macro F1 Score

epoch	1	2	3	4	5	6	7
fold 1	0.0271	0.2388	0.1981	0.2205	0.3434	0.2474	0.3119
fold 2	0.1333	0.0392	0.1414	0.1579	0.1927	0.20007	0.20004
fold 3	0.0090	0.2019	0.2264	0.2632	0.2251	0.2761	0.2712
fold 4	0.0857	0.1381	0.0916	0.1662	0.2217	0.3471	0.21677
fold 5	0.11698	0.19107	0.25591	0.2620	0.3943	0.30515	0.29464
fold 6	0.11324	0.25204	0.2210	0.14221	0.29556	0.288737	0.36270
fold 7	0.02060	0.1140	0.13503	0.151995	0.185981	0.23959	0.2252
fold 8	0.07574	0.07881	0.13889	0.15571	0.1960	0.22485	0.2552
avg	0.04395	0.15676	0.17606	0.18999	0.256875	0.266153	0.2672
std	0.04878	0.07662	0.05707	0.0506	0.078333	0.04754	0.05457

Table D.3: Cross-Entropy Model Micro F1 Score

D.2 Weighted Cross-Entropy Loss

epoch	1	2	3	4	5	6	7
fold 1	0.1116	0.1201	0.1020	0.0999	0.0988	0.1041	0.0940
fold 2	0.1041	0.1073	0.1041	0.1047	0.1041	0.1057	0.1020
fold 3	0.1057	0.0983	0.0967	0.0962	0.0940	0.0962	0.0919
fold 4	0.1057	0.1121	0.1041	0.0999	0.0983	0.0962	0.0983
fold 5	0.1148	0.1111	0.1036	0.1014	0.1068	0.1020	0.1052
fold 6	0.1132	0.1047	0.1025	0.1042	0.0994	0.094	0.0978
fold 7	0.101	0.0988	0.09615	0.09615	0.0946	0.0946	0.951
fold 8	0.1015	0.0967	0.1052	0.0994	0.0919	0.0999	0.0919
avg	0.10725	0.1062	0.1022	0.1003	0.0987	0.0991	0.0971
std	0.0053	0.00814	0.0029	0.0032	0.005	0.0045	0.0048

Table D.4: Weighted Cross Entropy Model Hamming Loss

epoch	1	2	3	4	5	6	7
fold 1	0.0430	0.1776	0.1631	0.2054	0.2635	0.2432	0.2587
fold 2	0.1331	0.0850	0.1341	0.1612	0.2202	0.2205	0.2275
fold 3	0.0520	0.1550	0.1785	0.2029	0.1888	0.2039	0.2185
fold 4	0.0449	0.0974	0.06309	0.1044	0.1458	0.1764	0.1370
fold 5	0.0510	0.1253	0.1984	0.2315	0.2511	0.2102	0.2511
fold 6	0.0470	0.1691	0.1941	0.1696	0.2871	0.2667	0.3418
fold 7	0.1162	0.1374	0.1977	0.1762	0.2797	0.2328	0.2536
fold 8	0.1214	0.1322	0.1657	0.1642	0.2300	0.1640	0.2532
avg	0.0761	0.1349	0.1618	0.1769	0.2333	0.2147	0.2427
std	0.0397	0.0325	0.0454	0.0381	0.0479	0.0339	0.0565

Table D.5: Weighted Cross Entropy Model Macro F1 Score

epoch	1	2	3	4	5	6	7
fold 1	0	0.2299	0.1751	0.2456	0.2809	0.2791	0.2927
fold 2	0.1245	0.0386	0.1330	0.1416	0.1949	0.2319	0.2014
fold 3	0	0.1688	0.2230	0.2391	0.2179	0.2591	0.2577
fold 4	0.0344	0.1534	0.0555	0.1403	0.2127	0.2493	0.1839
fold 5	0.0368	0.1447	0.2444	0.2872	0.3532	0.2680	0.3111
fold 6	0.0093	0.1929	0.2095	0.1826	0.3357	0.3102	0.3731
fold 7	0.1369	0.1686	0.2309	0.2073	0.3239	0.2626	0.2887
fold 8	0.0917	0.1234	0.1753	0.1688	0.2548	0.1941	0.2811
avg	0.0542	0.1525	0.1808	0.2016	0.2717	0.2568	0.2737
std	0.0558	0.0561	0.0623	0.0527	0.0610	0.0340	0.0603

Table D.6: Weighted Cross Entropy Model Micro F1 Score

D.3 Focal Loss

epoch	1	2	3	4	5	6	7
fold 1	0.1483	0.1297	0.1525	0.1084	0.1222	0.1307	0.1020
fold 2	0.1621	0.1775	0.1477	0.1169	0.1610	0.1371	0.1127
fold 3	0.1408	0.1477	0.1249	0.1212	0.1366	0.0887	0.1089
fold 4	0.1307	0.1504	0.1265	0.1254	0.1291	0.1217	0.1143
fold 5	0.1848	0.1570	0.1367	0.1538	0.1436	0.1522	0.1089
fold 6	0.1607	0.1260	0.1346	0.1004	0.1233	0.1068	0.1340
fold 7	0.1089	0.1164	0.1025	0.1068	0.1314	0.0998	0.1020
fold 8	0.1159	0.1613	0.0940	0.1036	0.1271	0.1068	0.1084
avg	0.1440	0.1458	0.1274	0.1171	0.1343	0.1180	0.1114
std	0.0253	0.0203	0.0204	0.0172	0.0128	0.0212	0.0101

Table D.7: Focal Model Hamming Loss

epoch	1	2	3	4	5	6	7
fold 1	0.1571	0.3256	0.3458	0.3669	0.3471	0.3787	0.4573
fold 2	0.2765	0.3401	0.1919	0.3055	0.3275	0.2653	0.3960
fold 3	0.2604	0.3292	0.3175	0.3128	0.3714	0.3235	0.3764
fold 4	0.2236	0.2660	0.2525	0.2689	0.2620	0.3297	0.3813
fold 5	0.2235	0.2993	0.3369	0.3264	0.3974	0.3550	0.3095
fold 6	0.2066	0.3447	0.3951	0.4133	0.4319	0.4149	0.4054
fold 7	0.1734	0.2663	0.3286	0.2768	0.3210	0.3273	0.3319
fold 8	0.2116	0.2753	0.2799	0.2937	0.3399	0.3316	0.3693
avg	0.2166	0.3058	0.3060	0.3205	0.3498	0.3408	0.3784
std	0.0398	0.0332	0.0628	0.0483	0.0515	0.0439	0.0451

Table D.8: Focal Model Macro F1 Loss

epoch	1	2	3	4	5	6	7
fold 1	0.1857	0.3732	0.3460	0.3852	0.4062	0.4769	0.4769
fold 2	0.3066	0.3622	0.2313	0.3362	0.3122	0.3021	0.3899
fold 3	0.3318	0.3462	0.3920	0.3900	0.4298	0.4408	0.4625
fold 4	0.2874	0.3447	0.3214	0.4028	0.3341	0.3974	0.4467
fold 5	0.3028	0.3751	0.4111	0.4223	0.4762	0.3811	0.3787
fold 6	0.2883	0.4329	0.4308	0.4787	0.4669	0.4581	0.4077
fold 7	0.1772	0.3178	0.3254	0.3268	0.3710	0.3409	0.3559
fold 8	0.2910	0.2931	0.3622	0.3432	0.4082	0.3777	0.4159
avg	0.2714	0.3556	0.3525	0.3857	0.4006	0.3881	0.4168
std	0.0573	0.0418	0.0628	0.0507	0.0587	0.0505	0.0423

Table D.9: Focal Model Macro F1 Score

Github Repositories

[1] URL: https://github.com/kadikraman/sorting.

[2] URL: https://github.com/darwinz/data-structures-and-algorithms.

[3] URL: https://github.com/silentapi/hackpack.

[4] URL: https://github.com/1995eaton/sorts.

[5] URL: https://github.com/HaysS/javascript-cs-fundamentals.

[6] URL: https://github.com/drminnaar/algorithms-and-data-structures.

[7] URL: https://github.com/shpotes/random-stuff.

[8] URL: https://github.com/alanwright/HackPack.

[9] URL: https://github.com/jeffrey-xiao/competitive-programming.

[10] URL: https://github.com/ashutoshbhadoria/data-structures-and-algorithms-javascript.

[11] URL: https://github.com/charulagrl/data-structures-and-algorithms.

[12] URL: https://github.com/seandroke/java-coursework.

[13] URL: https://github.com/varun93/javascript-challenges.

[14] URL: https://github.com/iiitv/algos.

[15] URL: https://github.com/hydencoder/Data-Structures.

[16] URL: https://github.com/travisvn/Data%5C_Structures%5C_And%5C_Algorithms.

[17] URL: https://github.com/ps0305/Javascript-Algorithms-And-Data-Structures.

[18] URL: https://github.com/aiy12/alg-library.

[19] URL: https://github.com/trevordjones/data%5C_structures%5C_and%5C_algorithms.

[20] URL: https://github.com/eyas-ranjous/sort-algorithms-js.

[21] URL: https://github.com/AbutoJoshua/libraryOfAlgorithms.

[22] URL: https://github.com/LordMathis/BinarySearchTree.

[23] URL: https://github.com/Quintec/AlgosAndData.

[24] URL: https://github.com/Ajantha8/SortingAlgorithms.

[25] URL: https://github.com/dacre-denny/js-avltree.

[26] URL: https://github.com/raavikr12c/Popular-Algorithms-in-Java.

[27] URL: https://github.com/ashish-chopra/Structures.

[28] URL: https://github.com/pomelo00o/TREE-COMPARISON.

[29] URL: https://github.com/miguendes/py-avl-tree.

[30] URL: https://github.com/RitRa/Algorithms-project-.

[31] URL: https://github.com/JuliaCollections/SortingAlgorithms.jl.

[32] URL: https://github.com/SvenWoltmann/sorting-algorithms-ultimate-guide.

[33] URL: https://github.com/tarcisio-marinho/sorting-algorithms.

[34] URL: https://github.com/Himanshu40/Sorting-Algorithm.

[35] URL: https://github.com/Thuva4/Algorithms.

[36] URL: https://github.com/diptangsu/Sorting-Algorithms.

[37] URL: https://github.com/gwtw/py-sorting/tree/master/sort.

[38] URL: https://github.com/play-Arena/sort-algorithms-with-java.

[39] URL: https://github.com/mazhar-h/sorting-algorithms.

[40] URL: https://github.com/Tom-Goring/Sorting-And-Searching-Algorithms.

[41] URL: https://github.com/gwtw/java-sorting.

[42] URL: https://github.com/indy256/codelibrary.

[43] URL: https://github.com/MightyPixel/algorithms.

[44] URL: https://github.com/davidrzs/DavesAlgorithmCollection.

[45] URL: https://github.com/aladdinpersson/Algorithms-Collection-Python.

[46] URL: https://github.com/jatin-773762/Algorithms.

[47] URL: https://github.com/realpacific/algorithms.

[48] URL: https://github.com/davestroud/Algorithm%5C_Fundamentals.

[49] URL: https://github.com/ayushs1ngh/Popular-Algorithms-in-Java.

[50] URL: https://github.com/koalamango/js-algorithms.

[51] URL: https://github.com/musikito/algorithms.

[52] URL: https://github.com/faustotnc/Algorithms.

[53] URL: https://github.com/crypticsy/Toolbox.

[54] URL: https://github.com/torressa/cspy.

[55] URL: https://github.com/shivajivarma/codebase.

[56] URL: https://github.com/huangsam/python-algorithms.

[57] URL: https://github.com/amilajack/js-algorithms.

[58] URL: https://github.com/Uditha95/AlgorithmHub.

[59] URL: https://github.com/thinkphp/seeds.py.

[60] URL: https://github.com/bzliu94/algorithms.

[61] URL: https://github.com/xingximing-xxm/Algorithm-Collection.

[62] URL: https://github.com/thecodershub/algorithms.

[63] URL: https://github.com/Hacktoberfest-Nepal/Algorithms.

[64] URL: https://github.com/nocotan/algorithm%5C_collection.

[65] URL: https://github.com/Algo-Phantoms/Algo-Tree.

[66] URL: https://github.com/michaelaah/Algorithms.

[67] URL: https://github.com/tulika-99/sorting-algorithms.

[68] URL: https://github.com/guyett92/algorithmic-programming.

[69] URL: https://github.com/sortingjs/sortingjs.

[70] URL: https://github.com/eugenmihailescu/sorting-algorithms.

[71] URL: https://github.com/benoitvallon/computer-science-in-javascript.

[72] URL: https://github.com/TracyGJG/All%5C_Sorts.

[73] URL: https://github.com/cschen1205/js-sorting-algorithms.

[74] URL: https://github.com/idosela/algorithms-in-javascript.

[75] URL: https://github.com/sgha10/sortem.

[76] URL: https://github.com/jongha/sort-js.

[77] URL: `https://github.com/jonmbake/javascript-sorting-algorithms`.

[78] URL: `https://github.com/make-github-pseudonymous-again/js-sorting`.

[79] URL: `https://github.com/jiayihu/pretty-algorithms`.

[80] URL: `https://github.com/jardrake/JavaScript-Sorting-Algorithms`.

[81] URL: `https://github.com/bcdxn/sort-util-js`.

[82] URL: `https://github.com/RefaelBeker7/Sorting-Algorithms-Python`.

[83] URL: `https://github.com/rekyungmin/python-sorting-algorithm`.

[84] URL: `https://github.com/alxojy/sorting-algorithms`.

[85] URL: `https://github.com/akanksha0411/Sorting-Algorithms-in-Python`.

[86] URL: `https://github.com/PabloAceG/sorting-algorithms`.

[87] URL: `https://github.com/Luke-zhang-04/Sorting-Algorithms`.

[88] URL: `https://github.com/Sayan-Paul/Sort-Library-in-Python`.

[89] URL: `https://github.com/akhof/Python-Sorting-Algorithms`.

[90] URL: `https://github.com/ztgu/sorting%5C_algorithms%5C_py`.

[91] URL: `https://github.com/rajko-z/sorting-algorithms`.

[92] URL: `https://github.com/Aly-Tomato/Sorting-Algorithms`.

[93] URL: `https://github.com/Zircoz/Sorting-Algorithms-in-Python`.

[94] URL: `https://github.com/vishnuvardhan-kumar/sorting-algorithms`.

[95] URL: `https://github.com/sreejithc321/Searching-and-Sorting-Algorithms`.

[96] URL: `https://github.com/MarcelKaemper/sortingAlgorithms`.

[97] URL: `https://github.com/arpol/spa`.

[98] URL: `https://github.com/pnikhare/sorting-algorithms-analysis`.

[99] URL: `https://github.com/aashrafh/sortizer`.

[100] URL: `https://github.com/chandantaneja/Sorting-Algorithms-1`.

[101] URL: `https://github.com/theroyakash/sorting-algos`.

[102] URL: `https://github.com/senavs/SortingAlgorithms`.

[103] URL: https://github.com/streethacker/Data-Structures-and-Algorithms-Using-Python.

[104] URL: https://github.com/ypedroo/sorting-algorithms.

[105] URL: https://github.com/vmdharan/Sorting-Algorithms.

[106] URL: https://github.com/burakguneli/sorting-algorithms-with-python.

[107] URL: https://github.com/osilkin98/SkipSort.

[108] URL: https://github.com/marie-at-mars/Sorting-Algorithms.

[109] URL: https://github.com/bai7th/Python-Common-Algorithm.

[110] URL: https://github.com/mihany/sorting-algorithms.

[111] URL: https://github.com/MartinThoma/algorithms.

[112] URL: https://github.com/Sriram-bb63/Sorting-algorithms.

[113] URL: https://github.com/subbarayudu-j/TheAlgorithms-Python.

[114] URL: https://github.com/Tomin-Joy/Sorting-Algorithms.

[115] URL: https://github.com/PuffyPotato/sorting-algorithms.

[116] URL: https://github.com/golharam/sorting%5C_algorithms%5C_python.

[117] URL: https://github.com/micromin/Sort-Algorithms-in-Java.

[118] URL: https://github.com/neerajd1/Sorting-Algorithms-in-Java-with-GUI.

[119] URL: https://github.com/melihtanriyakul/sorting-algorithms-in-java.

[120] URL: https://github.com/aokolnychyi/sorting-min.

[121] URL: https://github.com/otakaran/sorts-and-searches.

[122] URL: https://github.com/ashish2199/Sorting-Algorithms.

[123] URL: https://github.com/nishantroy/Data-Structures-and-Algorithms.

[124] URL: https://github.com/NunuM/Sorting-Algorithms.

[125] URL: https://github.com/satishkumarprasad/sorting-algorithms.

[126] URL: https://github.com/pledge/java-sorting.

[127] URL: https://github.com/Electr0d/SortingAlgorithms.

[128] URL: https://github.com/abdullahodibat/Sorting-Algorithms.

[129] URL: https://github.com/ErikBoesen/sort.

[130] URL: https://github.com/thedonat/Sorting-Algorithms-Comparison.

[131] URL: https://github.com/sj/TheAlgorithms-Java.

[132] URL: https://github.com/mvakili/all-sorts.

[133] URL: https://github.com/jipson7/sortingComparison.

[134] URL: https://github.com/lakshaygrover/Sorting-Algorithms.

[135] URL: https://github.com/Jeblas/CS-2331.

[136] URL: https://github.com/deepak-malik/Algorithms-In-Java.

[137] URL: https://github.com/showalter/sorting.

[138] URL: https://github.com/Hopson97/Sort-Algorithm-Visualiser.

[139] URL: https://github.com/calebkoy/sorting-algorithm-visualiser.

[140] URL: https://github.com/4math/sorting.

[141] URL: https://github.com/luigiselmi/algorithms.

[142] URL: https://github.com/YassinAJDI/algorithms-and-datastructures.

[143] URL: https://github.com/Dadadah/sorting-visualization.

[144] URL: https://github.com/mehmetpekdemir/Data-Structures-and-Algorithms.

[145] URL: https://github.com/patrickbucher/sorting.

[146] URL: https://github.com/akshaybahadur21/Sort.

[147] URL: https://github.com/jeffreybender/JavaSort.

[148] URL: https://github.com/nayuki/Sorting-algorithms-demo.

[149] URL: https://github.com/ramseth001/sorting.

[150] URL: https://github.com/pete314/algorithms-java.

[151] URL: https://github.com/arisath/Benchmarking-Sorting-Algorithms.

[152] URL: https://github.com/v-veryasov/interview-DataStructuresAndAlgorithmsJava.

[153] URL: https://github.com/HarderCode/Sorting-Algorithms-on-Java.

[154] URL: `https://github.com/ianramzy/sorting-comparison`.

[155] URL: `https://github.com/jasonsperske/SortLab.java`.

[156] URL: `https://github.com/iraghavr/interview-2`.

[157] URL: `https://github.com/Raymonoir/Number-sorting-algorithms-in-java`.

[158] URL: `https://github.com/sguttula/SortingAlgorithms`.

[159] URL: `https://github.com/jrquick17/java-sorting-algorithms`.

[160] URL: `https://github.com/Patepic/Sorting-Algorithms`.

[161] URL: `https://github.com/SeanCandon/Sorting-Algorithms-Comparison`.

[162] URL: `https://github.com/PPjamies/Sorting-Algorithms-App`.

[163] URL: `https://github.com/manojpawar94/sorting-and-search-algorithms-in-java`.

[164] URL: `https://github.com/MunzirAhmadh/Sorting-Algorithms`.

[165] URL: `https://github.com/Prateeksrt/JSort`.

[166] URL: `https://github.com/WWaldo/Sorting-Algorithms`.

[167] URL: `https://github.com/kdgyun/Sorting%5C_Algorithm`.

[168] URL: `https://github.com/kavirajkh/interview-1`.

[169] URL: `https://github.com/yogeshdarji/Searching-and-Sorting-Algorithms`.

[170] URL: `https://github.com/georgeberar/sorting-algorithms`.

[171] URL: `https://github.com/HadrienHachez/TheAlgorithms-Java-TEST`.

[172] URL: `https://github.com/Jojodicus/SortingAlgorithms`.

[173] URL: `https://github.com/TalhaKhamoor/SortingAlgorithms-JavaApplication`.

[174] URL: `https://github.com/georgecnicol/java-sort-practice`.

[175] URL: `https://github.com/natek1234/Gr12-11-Sorting-Algorithms-Java`.

[176] URL: `https://github.com/blaier23/Generic-Sorting-Algorithms`.

[177] URL: `https://github.com/SchnJulian/Sorting-Algorithms-CSharp-Java`.

[178] URL: `https://github.com/marioluan/java-sorting-algorithms`.

[179] URL: `https://github.com/argonautica/sorting-algorithms`.

[180] URL: https://github.com/devaaa225/SortingSimulator.

[181] URL: https://github.com/knrl/Java-Algorithms-and-Data-Structures.

[182] URL: https://github.com/shrushti2000/Data-Structures-and-Algorithms.

[183] URL: https://github.com/jeandersonbc/algorithms-and-ds.

[184] URL: https://github.com/AllenIrving/Sorting-Algorithms-in-Java.

[185] URL: https://github.com/techpanja/interviewproblems.

[186] URL: https://github.com/lasellers/SortAlgs.

[187] URL: https://github.com/AislingBoner/SortingAlgorithmBenchmarker.

[188] URL: https://github.com/jrquick17/java-sorting-algorithms.

[189] URL: https://github.com/rishisancheti/Sorting-Algorithms-OpenSource.

[190] URL: https://github.com/JoshuaKissoon/Merge-Sort.

[191] URL: https://github.com/faf0/LampSort.

[192] URL: https://github.com/Arham4/sorting-algorithms-analysis-se3345.

[193] URL: https://github.com/serveriev/sorting-algorithms.

[194] URL: https://github.com/dailong/Java-1.

[195] URL: https://github.com/ihsanizwer/Sorting-Algorithm-Simulator.

[196] URL: https://github.com/tuvo1106/python%5C_sorting%5C_algorithms.

[197] URL: https://github.com/vrompasa/sorting-demo.

[198] URL: https://github.com/JonComeau/Sorting-Algoritms.

[199] URL: https://github.com/SunderB/LDH-Build-2020-sorting-algorithms.

[200] URL: https://github.com/dsysoev/fun-with-algorithms.

[201] URL: https://github.com/pattersonrptr/sorting%5C_algorithms%5C_python.

[202] URL: https://github.com/sumedh151/Sorting-algorithms-in-python.

[203] URL: https://github.com/Surai98/sortingalgorithms.

[204] URL: https://github.com/awkbr549/sorting-algorithms.

[205] URL: https://github.com/mdisec/Comparison-Sorting-Algorithms.

[206] URL: https://github.com/negikaran7/Python-1.

[207] URL: https://github.com/okebinda/algorithms.python.

[208] URL: https://github.com/purple-worthy/TheAlgorithms-Python.

[209] URL: https://github.com/pyzh/Python-1.

[210] URL: https://github.com/chr12c/VisualSortingAlgorithms.

[211] URL: https://github.com/jsantana21/Sorting-Algorithm-Visualizer.

[212] URL: https://github.com/burningion/sorting-visualized.

[213] URL: https://github.com/FahadulShadhin/Sorting-Algorithms-Visualizer.

[214] URL: https://github.com/fazeVaib/SortingVisualizer.

[215] URL: https://github.com/alexmunoz502/Sorting-Algorithm-Visualizer.

[216] URL: https://github.com/deepak-malik/Data-Structures-In-Java.

[217] URL: https://github.com/sroopsai/HashMap-Implementation.

[218] URL: https://github.com/cfelde/BinaryOffheapHashMap.

[219] URL: https://github.com/ToxicXing/SimpleHashMap.

[220] URL: https://github.com/srflorea/HashMap-in-Java.

[221] URL: https://github.com/NipunSingh/HashMap-Implementation-In-Java.

[222] URL: https://github.com/Sanatan-Shrivastava/Algorithms-JAVA.

[223] URL: https://github.com/LukasPietzschmann/Hash-Map.

[224] URL: https://github.com/Gabriel-Cervo/Java-HashMap-Implementation.

[225] URL: https://github.com/manishpes/MyHashMap.

[226] URL: https://github.com/ksean/hash-map.

[227] URL: https://github.com/nikhilsu/Custom-Hash-Map.

[228] URL: https://github.com/contactsunny/HashMap%5C_Implementaion%5C_Java%5C_POC.

[229] URL: https://github.com/nicholassm/perfect-hashmaps.

[230] URL: https://github.com/Newbytee/nashmap.

[231] URL: https://github.com/mithun2595/kpcbHashMap.

[232] URL: https://github.com/Himanshu-Garg1/HashMapImplementation.

[233] URL: https://github.com/GodTamIt/java-hashmap.

[234] URL: https://github.com/intelie/tinymap.

[235] URL: https://github.com/norberte/ChainHashMap%5C_Implementation.

[236] URL: https://github.com/gaaiatinc/lru-cache-js.

[237] URL: https://github.com/furkankayar/Brent-sHashMethod.

[238] URL: https://github.com/Cultrarius/FastMap.

[239] URL: https://github.com/Mrignayni/Fixed-size-HashMap.

[240] URL: https://github.com/gouthamve/MVCCHM.

[241] URL: https://github.com/kbohinski/Generic-HashMap-Java.

[242] URL: https://github.com/joyfulmonster/concurrent-elastic-hashmap.

[243] URL: https://github.com/ThomasSkinner/Perfect-Spatial-Hashing-Java-Implementation.

[244] URL: https://github.com/giltene/PauselessHashMap.

[245] URL: https://github.com/giltene/PauselessHashMap.

[246] URL: https://github.com/JuhaS/SimpleMapStore.

[247] URL: https://github.com/vlsi/compactmap.

[248] URL: https://github.com/reines/persistenthashmap.

[249] URL: https://github.com/divyanshj16/hashmap.

[250] URL: https://github.com/vivekjustthink/WeakConcurrentHashMap.

[251] URL: https://github.com/abasko/linkedhashmap.

[252] URL: https://github.com/AngeloAvv/JBiHashMap.

[253] URL: https://github.com/raphw/weak-lock-free.

[254] URL: https://github.com/mp42/java-hashMap-X.

[255] URL: https://github.com/srflorea/HashMap-in-Java.

[256] URL: https://github.com/jaz888/Red-Black-Tree%5C_TreeMap%5C_Java.

[257] URL: https://github.com/zckeyser/all-the-structures.

[258] URL: https://github.com/rosberglinhares/DataStructuresAndAlgorithms.

[259] URL: https://github.com/saadati944/common%5C_algorithms%5C_and%5C_data%5C_structures.

[260] URL: https://github.com/NirmalSilwal/Data-Structure-and-Algorithm-Java-interview-kit.

[261] URL: https://github.com/inforkgodara/linear-search.

[262] URL: https://github.com/ayan-b/Linear-Search.

[263] URL: https://github.com/akshaybahadur21/Search.

[264] URL: https://github.com/ankit-hindustani/LinearSearch.

[265] URL: https://github.com/kpilszak/algorithms-implementation.

[266] URL: https://github.com/phishman3579/java-algorithms-implementation.

[267] URL: https://github.com/mohitsingla123/Data-Structure.

[268] URL: https://github.com/lifeparticle/Java-Algorithms-Implementation.

[269] URL: https://github.com/saihariG/Ultimate-Data-Structures-and-Algorithms-in-Java.

[270] URL: https://github.com/codegymdanang/CGDN-Algorithm.

[271] URL: https://github.com/Ashish138200/Data-Structures-and-Algorithm.

[272] URL: https://github.com/SrGrace/Algorithms%5C_Example.

[273] URL: https://github.com/navjindervirdee/data-structures.

[274] URL: https://github.com/AllAlgorithms/java.

[275] URL: https://github.com/bakhodir10/AlgoCS.

[276] URL: https://github.com/anis-ajmeri/Java-1.

[277] URL: https://github.com/dhruv007patel/Data-Structure-and-Algorithm-in-Java.

[278] URL: https://github.com/piersdb/Algorithms%5C_Example.

[279] URL: https://github.com/saranyanukala/Algorithms%5C_Example.

[280] URL: https://github.com/DeveloperKits/Algorithm-Basic.

[281] URL: https://github.com/j07nikita/Algorithms%5C_Example.

[282] URL: https://github.com/SVijayB/JavaSpace.

[283] URL: https://github.com/abhishekbisht1429/algorithms-and-data-structures.

[284] URL: https://github.com/AhmedSayedMansour/Data-Structure.

[285] URL: https://github.com/ErangaD/Algorithms%5C_Example.

[286] URL: https://github.com/panditakshay/JAVA-DS.

[287] URL: https://github.com/TamsilAmani/Algorithms%5C_Example.

[288] URL: https://github.com/AnugunjNaman/Algorithms.

[289] URL: https://github.com/Ashikpaul/Algorithms%5C_Example.

[290] URL: https://github.com/leomongeg/Algorithms%5C_Example.

[291] URL: https://github.com/callowidealist/Algorithms.

[292] URL: https://github.com/reach2arunprakash/Data-Structures.

[293] URL: https://github.com/jm8/all-the-algorithms.

[294] URL: https://github.com/toufiq-austcse/Algorithms%5C_Example.

[295] URL: https://github.com/bzdgn/data-structures-in-java.

[296] URL: https://github.com/dinofizz/hashmap-python.

[297] URL: https://github.com/mhw32/py-hashmap.

[298] URL: https://github.com/NikitaKhomenko/HashMap-implementation-in-python.

[299] URL: https://github.com/DavidSeptimus/python-hash-map.

[300] URL: https://github.com/adithyabsk/FixedHashMap.

[301] URL: https://github.com/avitaldrucker/hashmap.

[302] URL: https://github.com/lucasbrambrink/hash_map.

[303] URL: https://github.com/scotchka/hashmap.

[304] URL: https://github.com/christang/python-hashmap.

[305] URL: https://github.com/acucciniello/hashmap-python.

[306] URL: https://github.com/bharatnc/hash_map.

[307] URL: https://github.com/ncorbuk/Python-Tutorial---Dictionaries-Hash-Table-Hash-Map-Code-Walk-through-.

[308] URL: https://github.com/joykm88/hash-map.

[309] URL: https://github.com/flaurida/hashmap.

[310] URL: https://github.com/miko13/implement-hashmap.

[311] URL: https://github.com/moon05/hashmaps-python.

[312] URL: https://github.com/divyanshj16/hashmap.

[313] URL: https://github.com/lordzuko/DS-and-Algo-in-Python.

[314] URL: https://github.com/vandenheuvel/bimap.

[315] URL: https://github.com/marvincolgin/data-structures-and-algorithms.

[316] URL: https://github.com/loisgh/Hashmaps.

[317] URL: https://github.com/learning-zone/python-interview-questions.

[318] URL: https://github.com/kpapana/dataStructures.

[319] URL: https://github.com/anhhchu/pythonDSA.

[320] URL: https://github.com/Lakshya-Kejriwal/Hash-Map-Python.

[321] URL: https://github.com/AkshatM/NaiveHashmap.

[322] URL: https://github.com/Dholness2/hashmap-solution-.

[323] URL: https://github.com/NotMyThingy/HashMap.

[324] URL: https://github.com/ndevilla/hood.

[325] URL: https://github.com/nlarusstone/fixedsize-hashmap.

[326] URL: https://github.com/rhdmstjr/hash_adt.

[327] URL: https://github.com/abhinav-upadhyay/linear_hashing.

[328] URL: https://github.com/narendly/robinhoodhashing.

[329] URL: https://github.com/malewis5/Data-Structures-and-Algorithms-Python.

[330] URL: https://github.com/OmkarPathak/Python-Programs.

[331] URL: https://github.com/casthewiz/hashTable.

[332] URL: https://github.com/tinazheng/CuckooMap.

[333] URL: https://github.com/Logenleedev/Python-Self-learning-guide-including-algorithm-and-ML.

[334] URL: https://github.com/jushita/python-gibberish.

[335] URL: https://github.com/phantie/bound-sized-hash-map.

[336] URL: https://github.com/vbrazo/data-structures-archives.

[337] URL: https://github.com/AhmadTaj1754/ahmadHashMap.

[338] URL: https://github.com/LukasPietzschmann/Hash-Map.

[339] URL: https://github.com/NipunSingh/HashMap-Implementation-In-Java.

[340] URL: https://github.com/ToxicXing/SimpleHashMap.

[341] URL: https://github.com/srflorea/HashMap-in-Java.

[342] URL: https://github.com/d-kz/KPCB.

[343] URL: https://github.com/GodTamIt/java-hashmap.

[344] URL: https://github.com/mp42/java-hashMap-X.

[345] URL: https://github.com/manishpes/MyHashMap.

[346] URL: https://github.com/Gabriel-Cervo/Java-HashMap-Implementation.

[347] URL: https://github.com/contactsunny/HashMap_Implementaion_Java_POC.

[348] URL: https://github.com/intelie/tinymap.

[349] URL: https://github.com/norberte/ChainHashMap_Implementation.

[350] URL: https://github.com/massie/Cuckoo-hash-map.

[351] URL: https://github.com/hzulla/WeakValueHashMap.

[352] URL: https://github.com/Jeblas/CS-2331.

[353] URL: https://github.com/joyfulmonster/concurrent-elastic-hashmap.

[354] URL: https://github.com/HalilCan/custom_HashMap.

[355] URL: https://github.com/emd5/data-structures-and-algorithms-java.

[356] URL: https://github.com/mtumilowicz/java12-fundamentals-hash-map-implementation-workshop.

[357] URL: https://github.com/reines/persistenthashmap.

[358] URL: https://github.com/schwiet/hashmap.

[359] URL: https://github.com/TuurDutoit/spatial-hashmap.

[360] URL: https://github.com/hoverruan/jshashmap.

[361] URL: https://github.com/stephanebachelier/hashmapper.

[362] URL: https://github.com/dondish/donhash.

[363] URL: https://github.com/mootable/hashmap.

[364] URL: https://github.com/ridz0326/Hash-Map.

[365] URL: https://github.com/calebmadrigal/algorithms-in-python.

[366] URL: https://github.com/kylepw/hashtable.

[367] URL: https://github.com/Wikilicious/Python_Hash_Table_Dictionary_Implementation.

[368] URL: https://github.com/jhomswk/Hash_Table.

[369] URL: https://github.com/dankins/RoboHash.

[370] URL: https://github.com/huzmasood/hashtable.

[371] URL: https://github.com/shahidhj/Data-Structures-using-Python.

[372] URL: https://github.com/dsavg/MS-in-Data-Science-Projects.

[373] URL: https://github.com/gabrielegilardi/HashTable.

[374] URL: https://github.com/Dstar4/Hash-Tables.

[375] URL: https://github.com/felix021/pyshmht.

[376] URL: https://github.com/DeborahHier/CuckooHash.

[377] URL: https://github.com/Darthfett/Hashtable.

[378] URL: https://github.com/AhmetErenIncir/Custom-Hash-Table-Implementation.

[379] URL: https://github.com/LoyolaChicagoCode/cs2-lab12-hashtable-java.

[380] URL: https://github.com/mefekaskaya/Hash-Table---Java.

[381] URL: https://github.com/surajrampure/Data-Structures.

[382] URL: https://github.com/wcyuan/SimpleAlgos.

[383] URL: https://github.com/JohnKurlak/HashTable.

[384] URL: https://github.com/apvitek/JavaAlgorithmsAndDataStructures.

[385] URL: https://github.com/sharif-doghmi/linear-probing-hash-table.

[386] URL: https://github.com/sideffectjava/Hashtable-DS.

[387] URL: https://github.com/mmaklad/HashTable.

[388] URL: https://github.com/lenmorld/Flexible-Hash-Table-in-Java.

[389] URL: https://github.com/JsonChao/Awesome-Algorithm-Study.

[390] URL: https://github.com/halls7588/Data_Structures_in_15_Languages.

[391] URL: https://github.com/richardschris/algorithms.

[392] URL: https://github.com/kumailn/Algorithms.

[393] URL: https://github.com/bhavnavarshney/Algorithms-and-Data-Structures.

[394] URL: https://github.com/joeyajames/Python.

[395] URL: https://github.com/nishantroy/Data-Structures-and-Algorithms.

[396] URL: https://github.com/epalrov/java-collections.

[397] URL: https://github.com/sroopsai/HashMap-Implementation.

[398] URL: https://github.com/leocjj/sorting_algorithms.

[399] URL: https://github.com/VicodinAbuser/ZTM-DS-and-Algo-Python.

[400] URL: https://github.com/fuston05/cs-module-project-hash-tables_2.

[401] URL: https://github.com/AvivaShooman/CuckooHash.

[402] URL: https://github.com/GreatHayat/LinkedList-Python.

[403] URL: https://github.com/Sayan-Paul/Linked-list-in-python.

[404] URL: https://github.com/ashraf1lastbreath/LinkedList.

[405] URL: https://github.com/miracode/data-structures.

[406] URL: https://github.com/teroqim/linkedlist.

[407] URL: `https://github.com/harpreetsi/DataStructureImpl`.

[408] URL: `https://github.com/clair3st/Data-Structures`.

[409] URL: `https://github.com/grantgasser/linked-list`.

[410] URL: `https://github.com/gillkyle/python-linked-list`.

[411] URL: `https://github.com/kaustubhShetty/Data-Structures-and-Algo-in-Python`.

[412] URL: `https://github.com/OmkarPathak/Data-Structures-using-Python`.

[413] URL: `https://github.com/xscorp/linked-list-python`.

[414] URL: `https://github.com/melvinchia3636/Python-LinkedList`.

[415] URL: `https://github.com/ekand/singlylinkedlist`.

[416] URL: `https://github.com/christinlepson/LinkedList-Python`.

[417] URL: `https://github.com/BigWheel92/Singly-Linked-List-in-Python`.

[418] URL: `https://github.com/codebasics/data-structures-algorithms-python`.

[419] URL: `https://github.com/mr-CLK/LinkedList-Assignment`.

[420] URL: `https://github.com/bryanlimy/double-linked-list`.

[421] URL: `https://github.com/corinnelhh/data-structures`.

[422] URL: `https://github.com/ajakubek/python-llist`.

[423] URL: `https://github.com/M2skills/Linked-List-in-Python`.

[424] URL: `https://github.com/EthanSeaver/Python-Projects`.

[425] URL: `https://github.com/rgsoda/pypy-llist`.

[426] URL: `https://github.com/ivaste/Algorithms`.

[427] URL: `https://github.com/ArashMehraban/Singly-Linked-List-`.

[428] URL: `https://github.com/RohanBasavaraju/udacity-data-structures-algorithms-python`.

[429] URL: `https://github.com/lpapazow/python-linked-lists`.

[430] URL: `https://github.com/dotcs/py-linked-lists`.

[431] URL: `https://github.com/shahidhj/Data-Structures-using-Python`.

[432] URL: https://github.com/streethacker/Data-Structures-and-Algorithms-Using-Python.

[433] URL: https://github.com/maciejoliwa/linked-list-visualization.

[434] URL: https://github.com/EliHar-zz/List-ADT.

[435] URL: https://github.com/sanjusci/algos.

[436] URL: https://github.com/connorvhennen/Data_Structures_Algorithms_In_Python.

[437] URL: https://github.com/eclectic-boy/python-CDLLwS.

[438] URL: https://github.com/alicepham/python-singly-linked-list-sorting-algos.

[439] URL: https://github.com/azjkjensen/Unrolled-Linked-List.

[440] URL: https://github.com/TechnoMonoCo/Python-Doubly-Linked-List.

[441] URL: https://github.com/xavierwwj/linked-list-with-sort.

[442] URL: https://github.com/shreyasvedpathak/Data-Structure-Python.

[443] URL: https://github.com/pavitra14/LinkedListPy.

[444] URL: https://github.com/avitaldrucker/hashmap.

[445] URL: https://github.com/6thfdwp/pyalg.

[446] URL: https://github.com/harishvc/challenges.

[447] URL: https://github.com/hansrajdas/algorithms.

[448] URL: https://github.com/yogesh-chaudhari-77/DataStructures-Python.

[449] URL: https://github.com/subsr97/daily-coding-problem.

[450] URL: https://github.com/tirthshah147/Data_Structures_in_Python.

[451] URL: https://github.com/balandhanka/Data-Structures-using-Python.

[452] URL: https://github.com/arsummers/python-data-structures-and-algorithms.

[453] URL: https://github.com/Abdullahi-a-hussein/ADT.

[454] URL: https://github.com/ehsanyousefzadehasl/PyDS.

[455] URL: https://github.com/suhassrivats/Data-Structures-And-Algorithms-Implementation.

[456] URL: https://github.com/BFriedland/data-structures.

[457] URL: `https://github.com/saketvikram/algorithms-ds-in-python`.

[458] URL: `https://github.com/OmkarPathak/Data-Structures-using-Python`.

[459] URL: `https://github.com/nikhilbalwani/xor-linked-list`.

[460] URL: `https://github.com/sakebow/python-linklist`.

[461] URL: `https://github.com/munna94/Python_Data_Structures`.

[462] URL: `https://github.com/ppant/DS-Algos-Python`.

[463] URL: `https://github.com/devinus/winked`.

[464] URL: `https://github.com/anubhavshrimal/Data-Structures-Algorithms`.

[465] URL: `https://github.com/TsHristov/Data-Structures-And-Algorithms`.

[466] URL: `https://github.com/joaolcaas/py-data-structure`.

[467] URL: `https://github.com/icarrera/data-structures`.

[468] URL: `https://github.com/joykm88/deque-bag-adt`.

[469] URL: `https://github.com/ysong-2018/Data-Structures-and-Algorithms-in-Python`.

[470] URL: `https://github.com/ramanaditya/data-structure-and-algorithms`.

[471] URL: `https://github.com/szaghi/single-linked-list`.

[472] URL: `https://github.com/the-algorithms/javascript-data-structures-algorithms`.

[473] URL: `https://github.com/divya-nk/py-ds-algorithms`.

[474] URL: `https://github.com/farooq98/HashTable`.

[475] URL: `https://github.com/LZQthePlane/Data-Structure-Algorithm-python`.

[476] URL: `https://github.com/1996sajal/Data-Structure-in-python`.

[477] URL: `https://github.com/acucciniello/hashmap-python`.

[478] URL: `https://github.com/liamgbs/pyADS`.

[479] URL: `https://github.com/omarsayed7/Data-Stracture-with-Python`.

[480] URL: `https://github.com/habiburrahman-mu/DS-Algo-Python`.

[481] URL: `https://github.com/VineetPrasadVerma/Udacity-DataStructures-And-Algorithms-In-Python`.

[482] URL: `https://github.com/exercism/problem-specifications`.

[483] URL: `https://github.com/hanhwi/python-datastructure`.

[484] URL: `https://github.com/BazNick/Doubly-Linked-List`.

[485] URL: `https://github.com/ghoshkaj/datastructures`.

[486] URL: `https://github.com/albert-espin/uno-linked-list/tree/master/Code`.

[487] URL: `https://github.com/rahul-tuli/DataStructuresForPython`.

[488] URL: `https://github.com/sukritishah15/DS-Algo-Point`.

[489] URL: `https://github.com/mick2004/DS`.

[490] URL: `https://github.com/kavdi/data-structures`.

[491] URL: `https://github.com/naagarjunsa/data-structures-python-primer`.

[492] URL: `https://github.com/rajithst/Python-Data-Structures-and-Algorithms`.

[493] URL: `https://github.com/mkirchner/linked-list-good-taste`.

[494] URL: `https://github.com/anhhchu/pythonDSA`.

[495] URL: `https://github.com/btj/ogp-notes/blob/master/iterators.md`.

[496] URL: `https://github.com/akiljames83/Pypeline`.