NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4021/CZ4034: Information Retrieval

Course Project Report

Group 8

Name	Matriculation	Contribution
Jared Nathaniel Chan Weng Wai	U2022954B	Classification and Report write up
Liau Zheng Wei	U2222032K	Classification and Indexing
Toh Yong Li	U2021290D	Web Crawling and UI development
Beh Ming Jun	U2022747E	Classification and Report write up
Ooi Yi Hang	U2022327K	Classification and Video
Wentai Nan	U2022249B	Classification and Slides

Content Page

1. Introduction

1.1 Selection of Topic

In recent years, the volume of financial news available to investors has surged exponentially, making it increasingly challenging for investors to learn from the vast amount of information to gauge the market sentiment towards different stocks. The objective of our project is to build an opinion search engine that allows users to explore and analyse the sentiment of financial news related to different stocks. By selecting a stock of interest, users can obtain a detailed sentiment analysis, quantifying the percentage of positive and negative opinions in the news. This engine will enable investors to make more informed decisions based on the prevailing sentiment in the news.

2. Crawling of News Text Data

Initially, we had the idea to scrape reddit and also new websites and combine the results together. Specifically, we wanted to scrap the daily discussions posts on r/stocks. However, we soon found out that there were many issues involving this task. Firstly, on some trading days, there were no daily discussions posts. Secondly, on some days, the number of comments related to the few stocks that we chose were very few. Thirdly, due to the structure of reddit and the culture of users hijacking the highest upvoted posts to comment, the value of main posts drops significantly after a certain number of upvotes, rendering it difficult to determine which comments should be used. Fourthly, we could not wrap our head around how we would combine these two types of web scrape data together. Lastly, using the PRAW api to scrap reddit, we found that the api was crashing due to the scraping limit. Due to the combination of these factors, we decided to drop the idea of scraping reddit and stuck to scraping websites.

2.1 Methodology of Web Scraping

In this project, we detail the approach employed to scrape news articles associated with a designated list of stocks from Yahoo News. The process relies on several Python libraries, including requests for making HTTP requests, BeautifulSoup for parsing HTML, pandas for

data manipulation, re for regular expression operations, csv for data storage, and datetime for managing timestamps. Central to the methodology are key functions designed for specific tasks. The get_article(card) function extracts essential details such as headlines, sources, posting times, descriptions, and links from news article cards on Yahoo News. Meanwhile, the get_the_news(search) function conducts searches for news articles related to specific stock symbols, retrieves them, and organises them into a structured DataFrame. Additionally, the convert_relative_time_to_date(relative_time) function plays a critical role in converting relative posting times (e.g., "2 hours ago", "1 day ago") into actual dates, ensuring accurate temporal representation.

The scraping process unfolds systematically, beginning with the definition of a list of target stocks for which news articles are sought. Subsequently, an iterative process is initiated, wherein each stock undergoes the scraping procedure. Within this loop, the get_the_news(stock) function is invoked to retrieve pertinent news articles for the current stock. These articles are then subjected to a conversion process, where relative posting times are transformed into actual dates through the convert_relative_time_to_date() function. Finally, the extracted data is stored in CSV files, each named after the corresponding stock symbol, facilitating organised storage and subsequent analysis.

2.2 Analysis of Crawled Data in Text Corpus

2.2.1 Type of information to retrieve

In our web scraping, we decided to retrieve the following data:

1. Headline of the news article
2. Source of the news article
3. Date Posted of the news article
4. A brief description of the first paragraph of the article
5. Link to the news article website

Data	Purpose	Data Type
Headline	Display, Sentiment Model Testing	String

Source	Display	String
Posted	Display	Date
Description	Display	String
Link	Display	String

Table 1: Description of Data field of Crawled Corpus

The data we decided to scrap mainly focuses on 5 stocks:

1. Amazon(AMZN)
2. Google(GOOG)
3. Nvidia(NVDA)
4. Apple(AAPL)
5. Microsoft(MSFT)

2.2.2 Number of records, words and types in the corpus

Due to the large amount of data that has to be scraped, we did the web scraping in batches. Then we combined the datasets together and cleaned the crawled data. First, we dropped duplicate data between rows but we still found duplicates in the column headers “Headline” and “Link”, which we later dropped as well. This ensures that the user querying for results will not receive articles with the same headline or link, thereby improving the user experience.

Number of Records	Number of Words
10,916	484,908

Table 2: Number of Records and Words of corpus

2.3 Online Sentiment Data

2.3.1 What are the datasets used and where were they obtained from?

To train the model we used to perform sentiment prediction, we used online sentiment benchmarks from huggingface and kaggle. The dataset from huggingface is: zeroshot/twitter-financial-news-sentiment · Datasets at Hugging Face while the kaggle

datasets are: Stock-Market Sentiment Dataset (kaggle.com) and https://www.kaggle.com/datasets/ankurzing/sentiment-analysis-for-financial-news.

After obtaining the datasets from the above links, we performed data preprocessing by analysing the structure of the columns of the csv and rearranging and relabelling them so that all the datasets match. Within the 'Sentiment' columns of each of the datasets, there were different representations of the same value. For example, the sentiment 'negative' can be represented in this form in one dataset, as -1 in another dataset and as 0 in another dataset. After which, we removed duplicates within the 'Headline' column.

2.3.2 Data type, number of words and records

Data	Purpose	Data Type
Headline	Model Sentiment Training	String
Sentiment	Model Sentiment Training	Integer

Table 3: Description of Data Field of Online Sentiment Data

Number of Records	Number of words
22935	395,427

Table 4: Number of Records and Words of Online Sentiment Data

3. Indexing and Querying

3.1 Data Indexing and Querying using Apache Solr

This project utilised Apache Solr as a backend to index and store the data. Solr is an open-source enterprise-search platform, written in Java.

3.1.1 Data Fields

The table showcases the fields, description of the fields and an example of data in the field.

Field	Field Description	Example
Headline	Headline of news article	“Amazon.com, Inc. (NASDAQ:AMZN) Stake Raised by Independent Wealth Network Inc.”
Source	Source of news article	“ETF DAILY NEWS”
Posted	Date the news article was posted	“30/3/2024”
Description	Brief introduction to the news article	“Independent Wealth Network Inc. boosted its holdings in Amazon.com, Inc. (NASDAQ:AMZN) by 7.5%...”
Link	URL link of the news article	“https://www.etfdailynews.com/2024/03/30/amazon-com-inc-nasdaqamzn-stake-raised-by-independent-wealth-network-inc/”
Sentiment	Subjectivity or polarity of headline	“Positive”

Table 5: Description of Data Fields in Solr with examples

3.1.2 Solr Configurations

Solr includes features like full-text search, real-time indexing, tokenization, stop-word removal, stemming, and spell checking. Given that Solr has these features already in place, we decided to not make any alterations to it. We added some synonyms to map the stock ticker symbol to the company name.

3.1.3 Data Indexing in Solr

To index the data into Solr, we created a python script csvdataimport.py which utilises pysolr and csv modules to interact with Solr and for CSV file handling. A connection is created to the Solr server with ‘always_commit=True’ parameter ensuring that changes are committed immediately after each document addition. Each row of the CSV file is read as a dictionary representing a Solr document and the keys are column headers. The keys of the dictionary correspond to the fields in the Solr schema. The Solr documents are then added to the Solr index. Through this method, we have tokenized and indexed the CSV content.

```python
import csv
import pysolr

# Connect to solr
solr = pysolr.Solr('http://localhost:8983/solr/stocks_core2', always_commit=True)

print("Connected")

# Read CSV and index documents
with open('C:/Users/tohyo/Downloads/solr-9.5.0/solr-9.5.0/server/solr/stocks_core2/conf/combined_csv.csv',
          'r', encoding='utf-8') as csvfile:
    print("reading")
    reader = csv.DictReader(csvfile)

    # Create a dictionary to store columns
    columns = {}
    count=0
    # Iterate over rows
    for row in reader:
        # Format data for Solr
        solr_doc = {
            'Index':row.get('Index'),
            'Headline':row.get('Headline'),
            'Source':row.get('Source'),
            'Posted':row.get('Posted'),
            'Description':row.get('Description'),
            'Link':row.get('Link'),
            'Sentiment':row.get('FINAL Sentiment'),
        }
        count+=1
        print(count)
        #Index document
        solr.add([solr_doc])

print("Import complete")
```

Figure 1: Add documents to Solr

3.1.4 Querying in Solr

Querying in Solr is a crucial aspect of retrieving relevant information from indexed documents. By crafting precise queries, users can extract valuable insights from the vast pool of indexed data. In this section, we delve into a practical example of querying in Solr using the Python pysolr library.

In our workflow, we first establish a connection to the Solr instance hosted at http://localhost:8983/solr/stocks_core2. Subsequently, a custom query is constructed using parameters like custom_query, start_date, and end_date to tailor the search criteria. The solr.search() method then executes the query, retrieving documents that satisfy the specified conditions. These retrieved documents are iterated over, with pertinent information such as the headline being accessed for further processing. This systematic approach allows us to

efficiently retrieve and analyse relevant data from the Solr index, facilitating subsequent actions based on the retrieved information. In Solr's configuration, we have added a list of stopwords based on nltk's english stopwords list. Furthermore a list of synonyms mapping company names to tickers have been added. These enhancements are applied to the query for greater convenience.

```
import pysolr

# Connect to the Solr instance
solr = pysolr.Solr('http://localhost:8983/solr/stocks_core2', always_commit=True)

# Execute the Solr query
response = solr.search('Headline:\"'+ custom_query + "\" AND Posted:[\"" + start_date
                       + "T00:00:00Z\"" + " TO \"" + end_date + "T23:59:59Z\"]",
                       **{'start': 0, 'rows': 20000, 'sort':'Posted ' + asc})

# Iterate over the query results
for hit in list(response):
    # Access relevant fields from each document
    headline = hit['Headline'][0]
    # Further processing of retrieved data...
```

Figure 2: Solr query

3.2 User Interface for Querying Crawled Data

3.2.1 Overview

The User Interface for Querying Crawled Data provides users with an intuitive platform to interact with and retrieve information from a crawled dataset. The interface comprises several components, including a sidebar for navigation, search functionality, filtering options, and result display sections as shown in Figure 3 below.

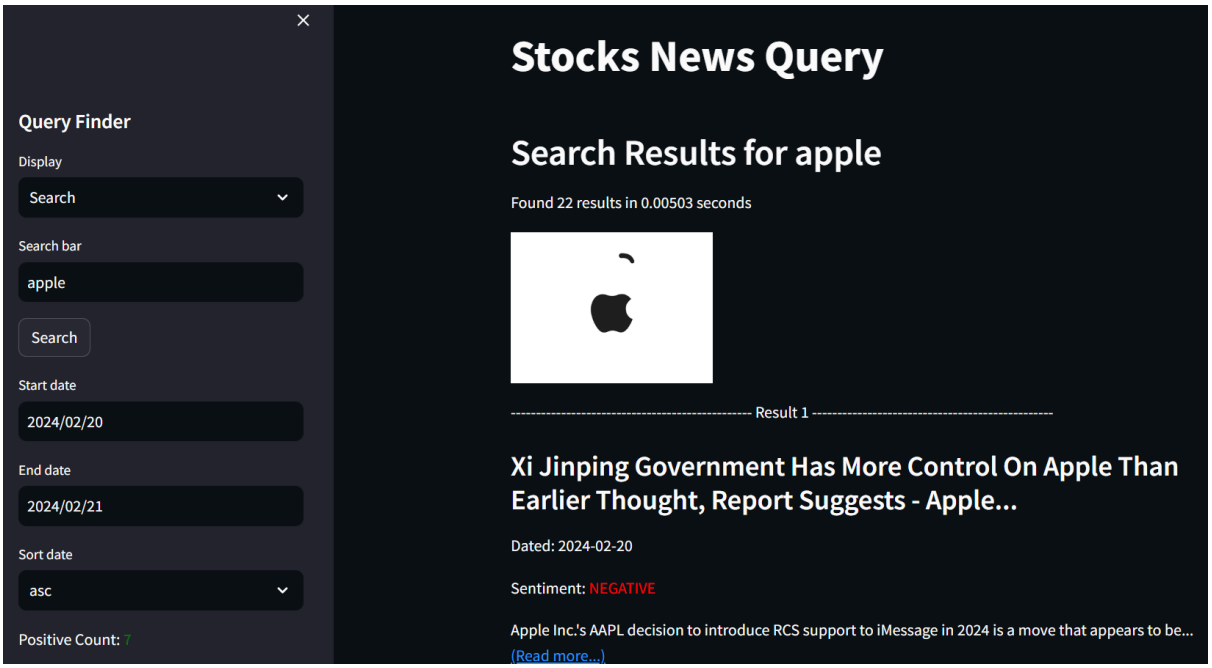

Figure 3: UI Overview

3.2.2 Sidebar Navigation

The sidebar offers options to choose between 'Search' or 'About Database'. Users can easily switch between these options to perform searches or access information about the database as shown in Figure 4 below.

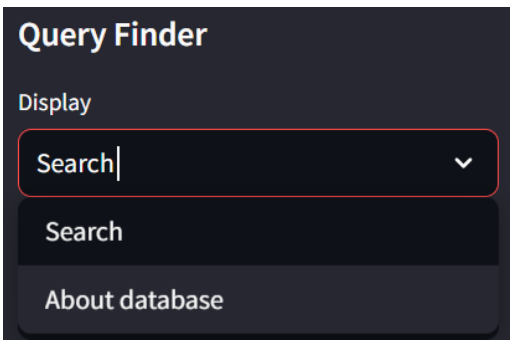

Figure 4: Display options

3.2.3 Search Functionality

The interface includes a search bar where users can input their query. Users have the flexibility to filter search results based on date, refining their queries to specific time frames as shown in Figure 5 below.

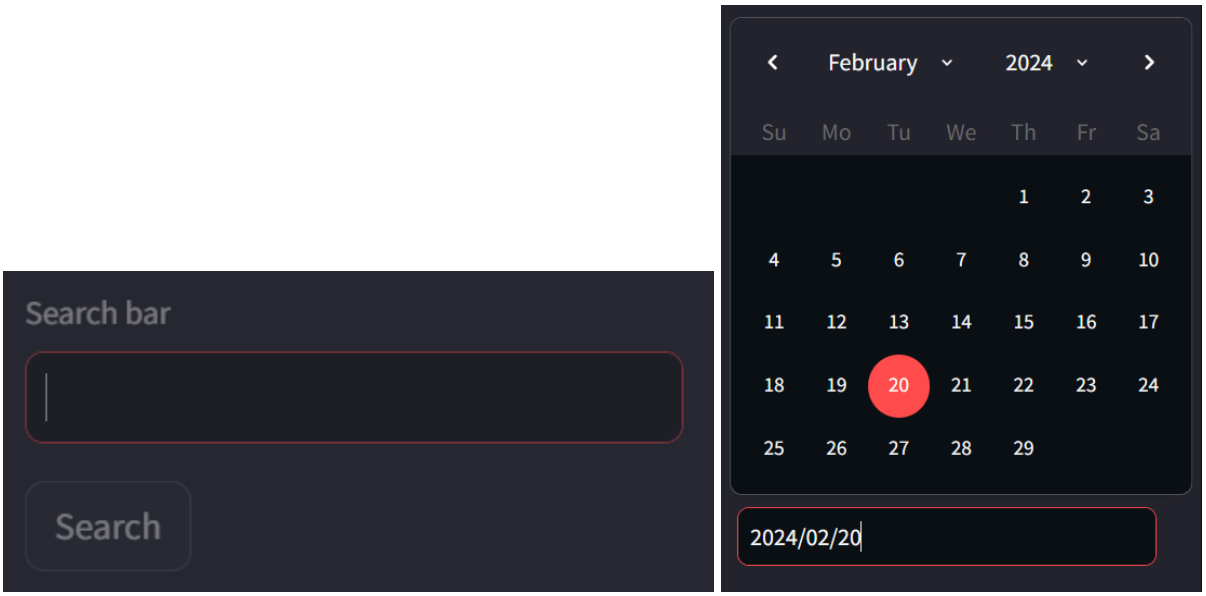

Figure 5: Searchbar (left) and Date Picker (right)

3.2.4 Sorting Options

Users can choose to sort search results by date in ascending or descending order.
This feature enhances user experience by allowing them to organize search results according to their preferences as shown in Figure 6 below.

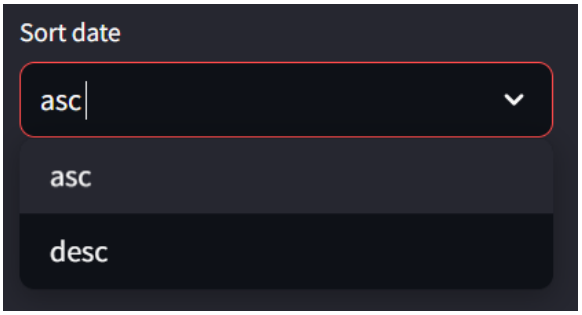

Figure 6: Sort date options

3.2.5 Sentiment Analysis

The interface provides counts for positive, negative, and neutral sentiments related to the search query. This sentiment analysis feature offers users insights into the overall sentiment distribution of the search results as shown in Figure 7below.

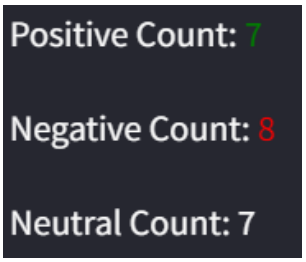

Figure 7: Sentiment Count

3.2.6 Main Page Display

Upon performing a search, the main page displays details of the query, including the queried term and any applied filters. Users can view the total number of results found and the search time, enabling them to assess the efficiency of their queries. An image related to the queried term is displayed to provide visual context. Search results are presented in a structured format, allowing users to easily navigate through the retrieved information as shown in Figure 8 below.

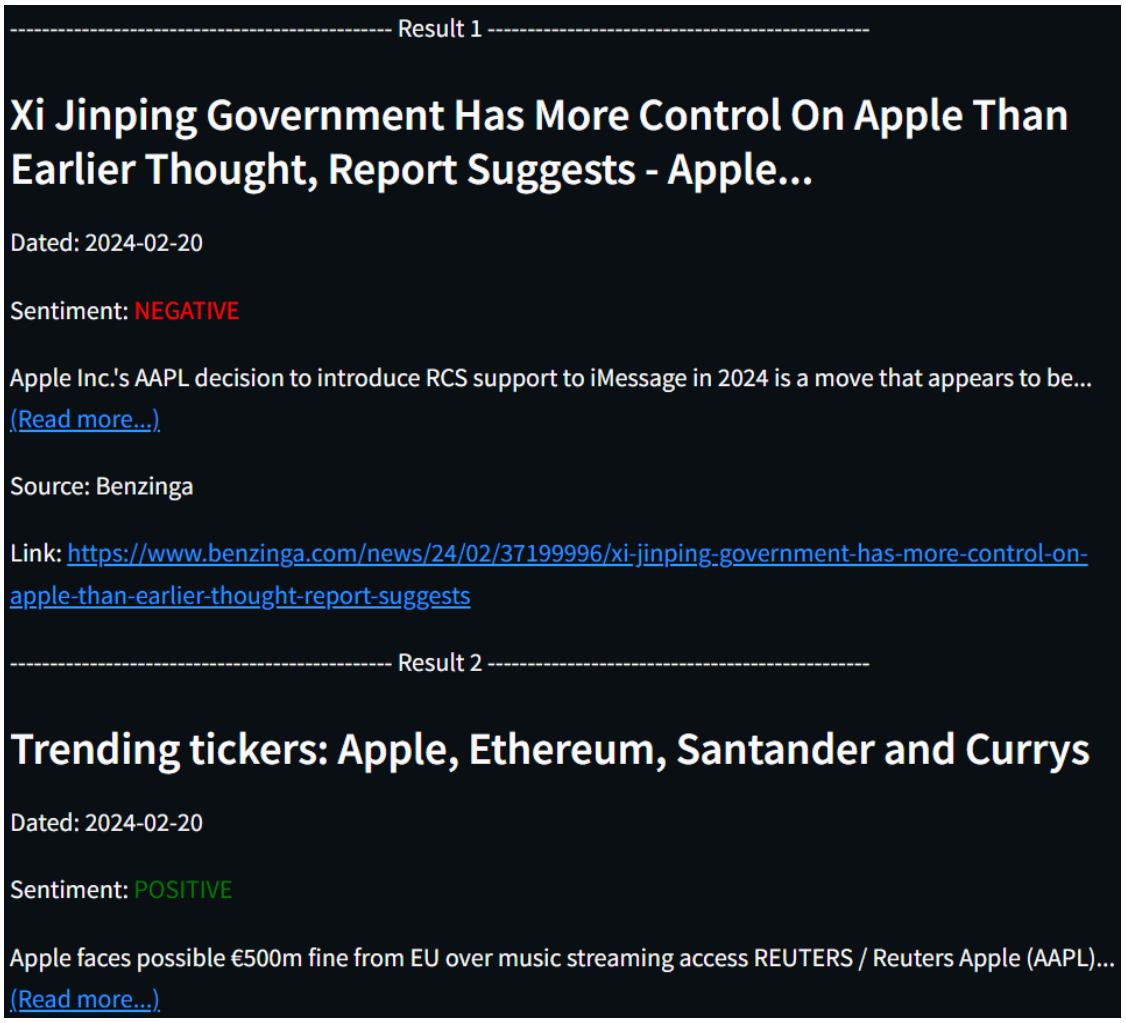

Figure 8: Search Result UI

3.2.7 'About Database' page Display

The 'About Database' page provides users with a holistic understanding of the indexed database through insightful visualisations. Users can explore the daily document count by term stacked bar chart to track document occurrences over time, while the word cloud offers a visual representation of prominent terms within the dataset. The pie chart breaks down document distribution across categories or topics, aiding in understanding the database's composition. Additionally, the sentiment plot visualises sentiment analysis results, allowing users to grasp the overall sentiment landscape of the database. Together, these visualisations enable users to extract valuable insights and make informed decisions based on the indexed dataset's content and sentiment distribution as shown in Figure 9 below.

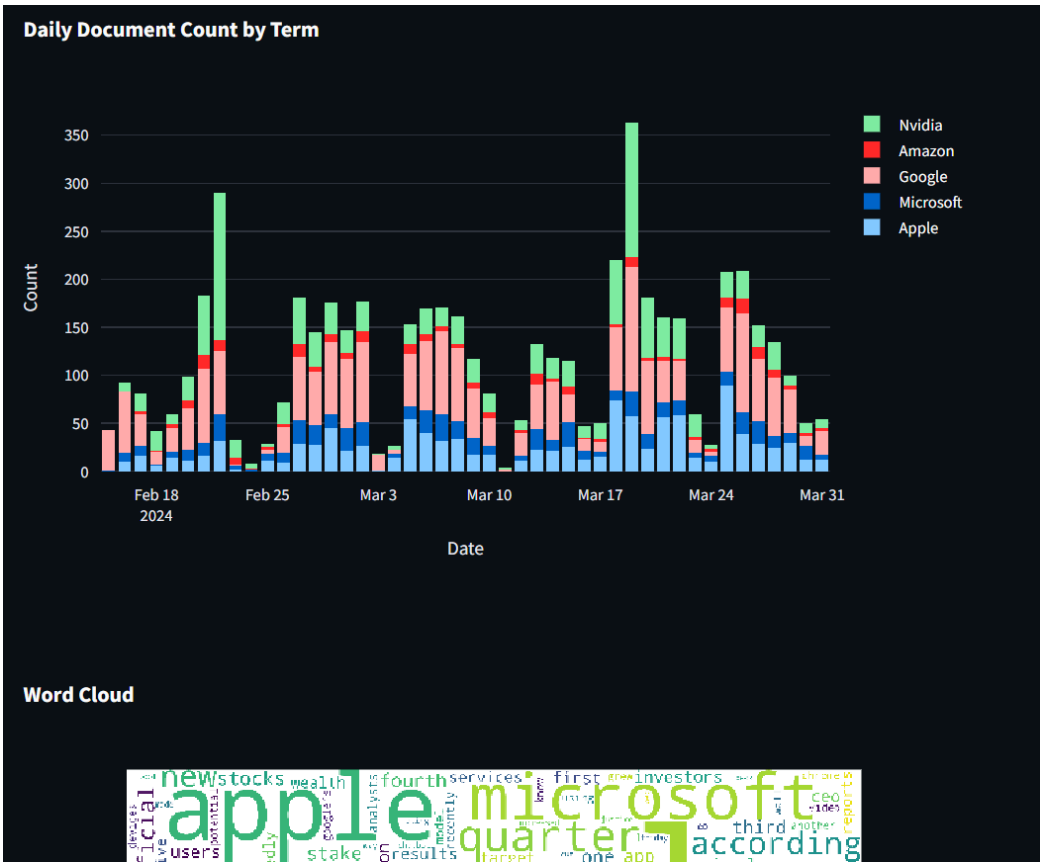

Figure 9: About Database UI

3.3 Query Results and Performance

This report analyses the query results for the terms "apple," "google," "nvidia," "amazon," and "microsoft" in the context of a search application. The report evaluates the relevance and performance of the query results to provide insights into the effectiveness of the search functionality.

3.3.1 Query Results Analysis

- **Query Terms:** The queries were performed for the following terms: "apple," "google," "nvidia," "amazon," and "microsoft."
- **Relevance:** The relevance of the query results was assessed based on the degree to which they matched the user's search intent.
- **Performance:** The performance of each query was measured in terms of query processing time, which ranged from 0.027 to 0.19247 seconds. Notably, the query for

"amazon" had the fastest processing time at 0.027 seconds, while the query for "google" had the longest processing time at 0.19247 seconds.

3.3.2 Observations and Insights

- The query results appeared to be relevant for all terms, with documents related to the respective companies being prominently featured.
- The processing time varied across different queries, with some queries being processed faster than others. The faster processing times may indicate more optimised indexing or caching mechanisms for certain terms.
- Despite the differences in processing time, the overall performance of the search functionality remained acceptable, with all queries returning results within a reasonable timeframe.

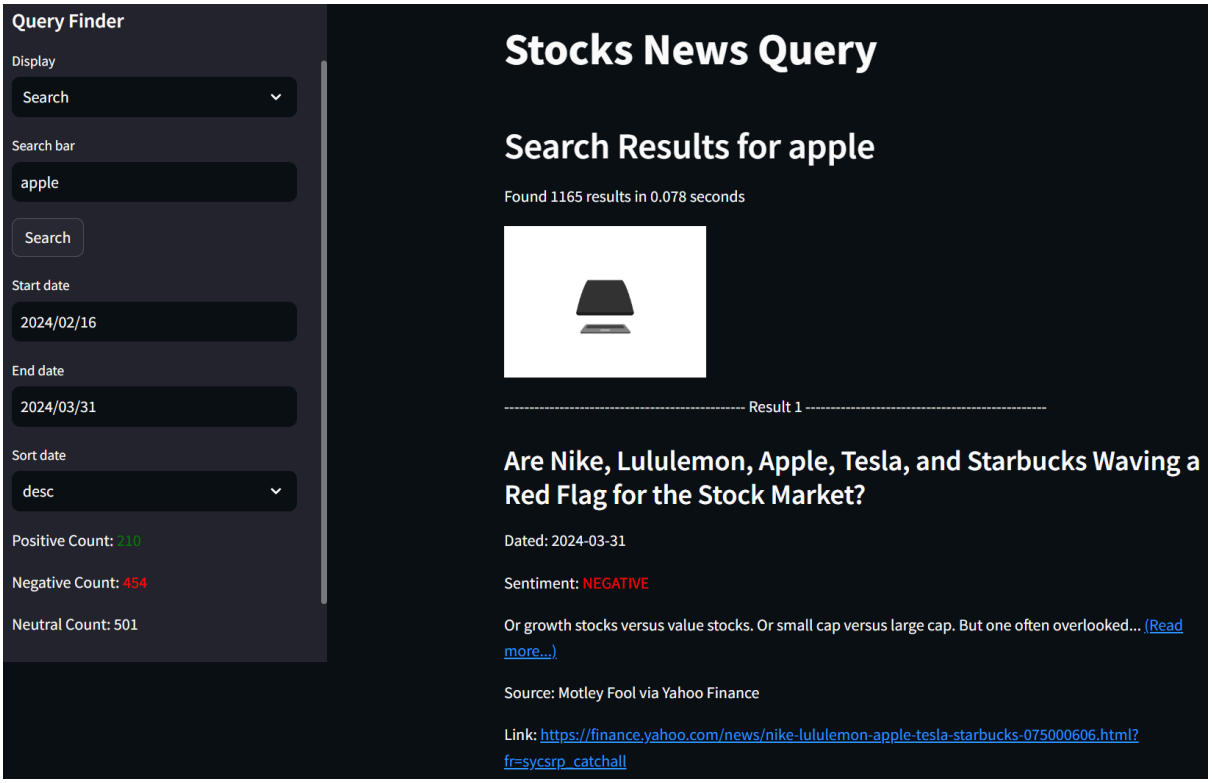

Figure 10: Query 1

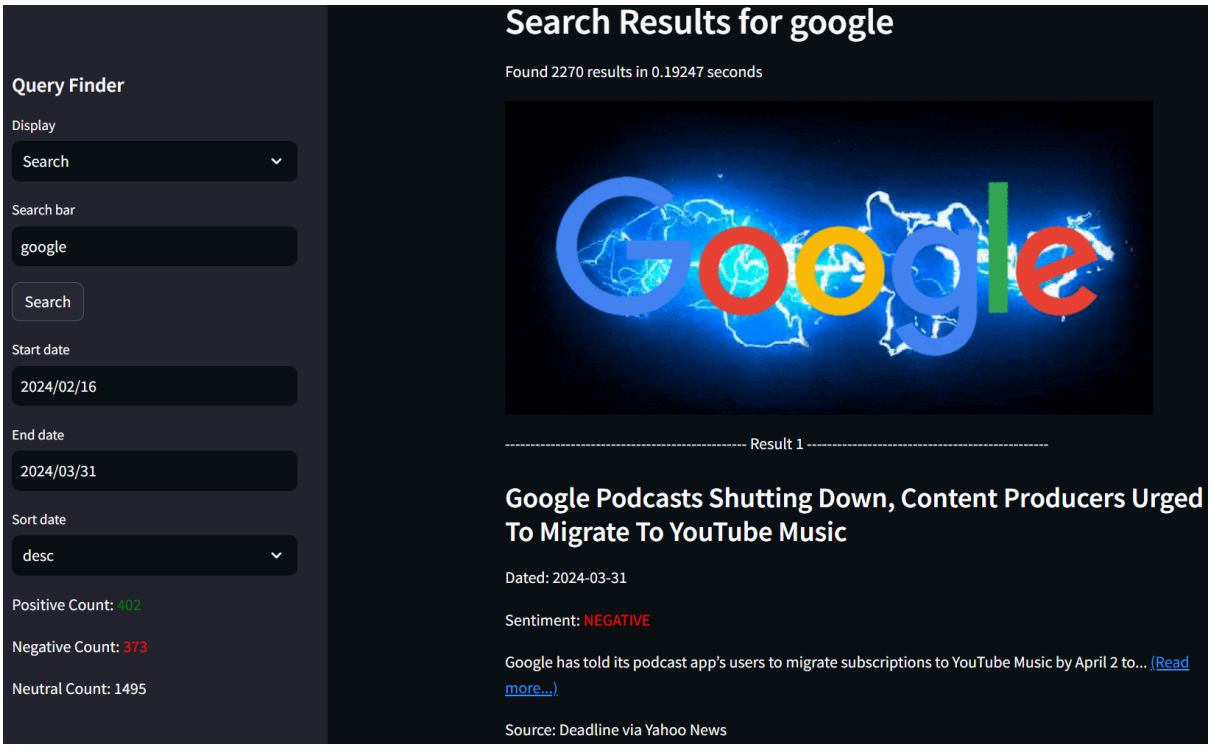

Figure 11: Query 2

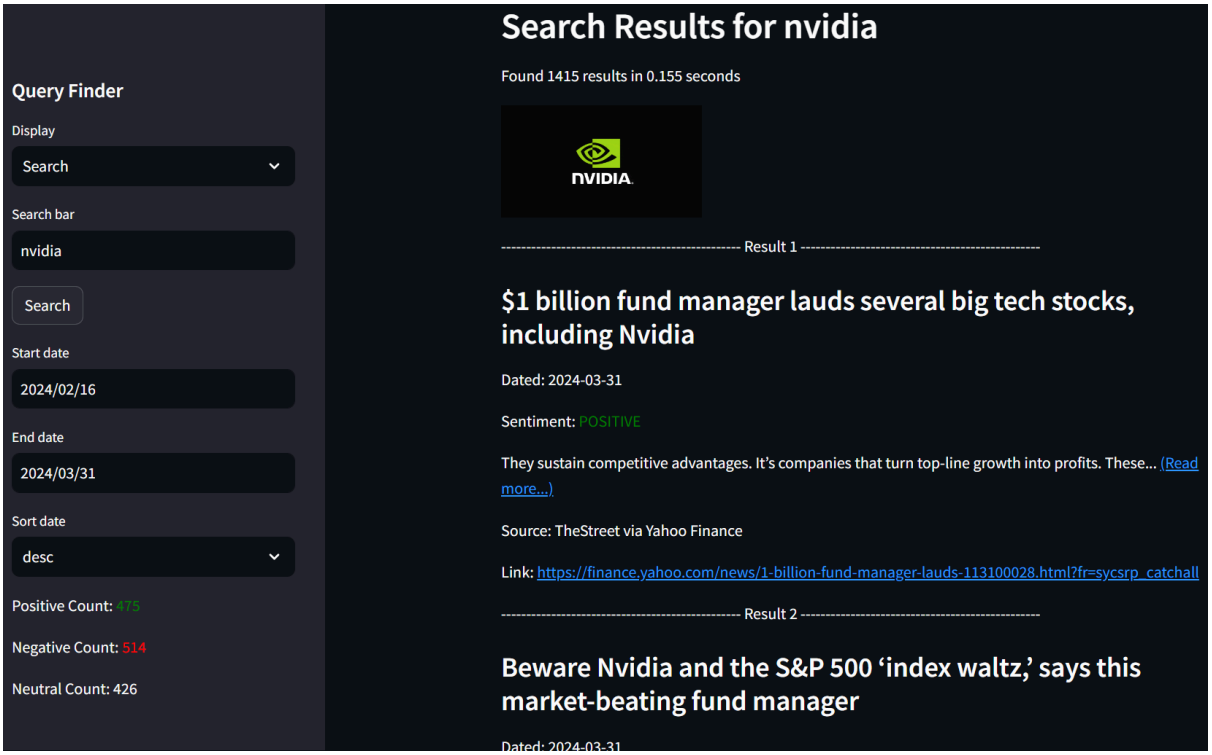

Figure 12: Query 3

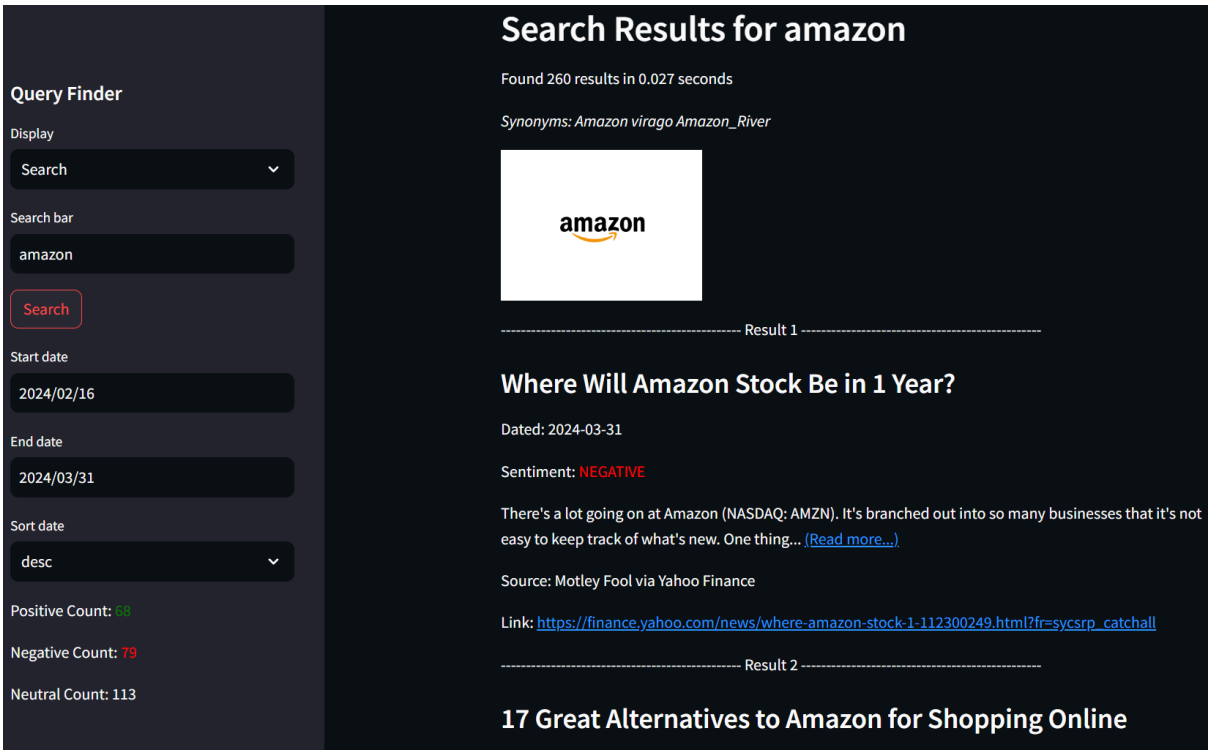

Figure 13: Query 4

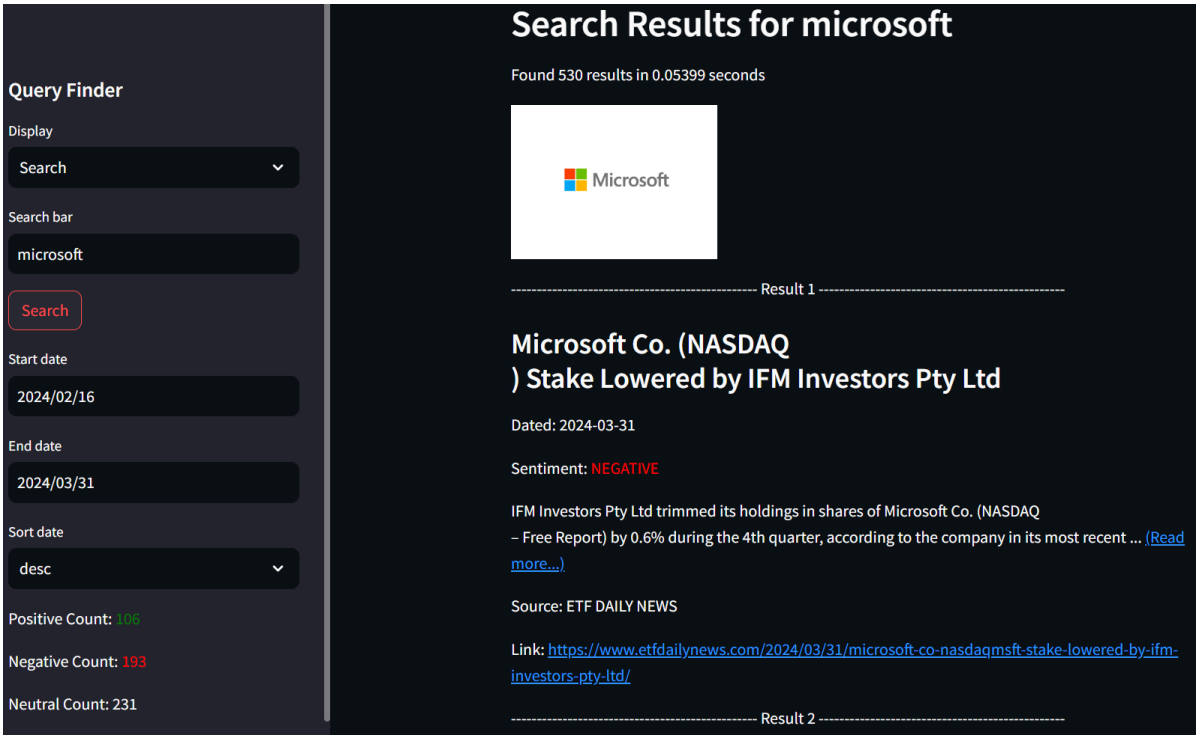

Figure 14: Query 5

3.4 Innovations for enhancing Querying

3.4.1 Multilingual Search

We included the ability for users to search for the different stock in different languages, namely Chinese. When users search for the stock name in Chinese, for example “苹果” is apple in Chinese and it will be able to display all the results regarding to Apple.

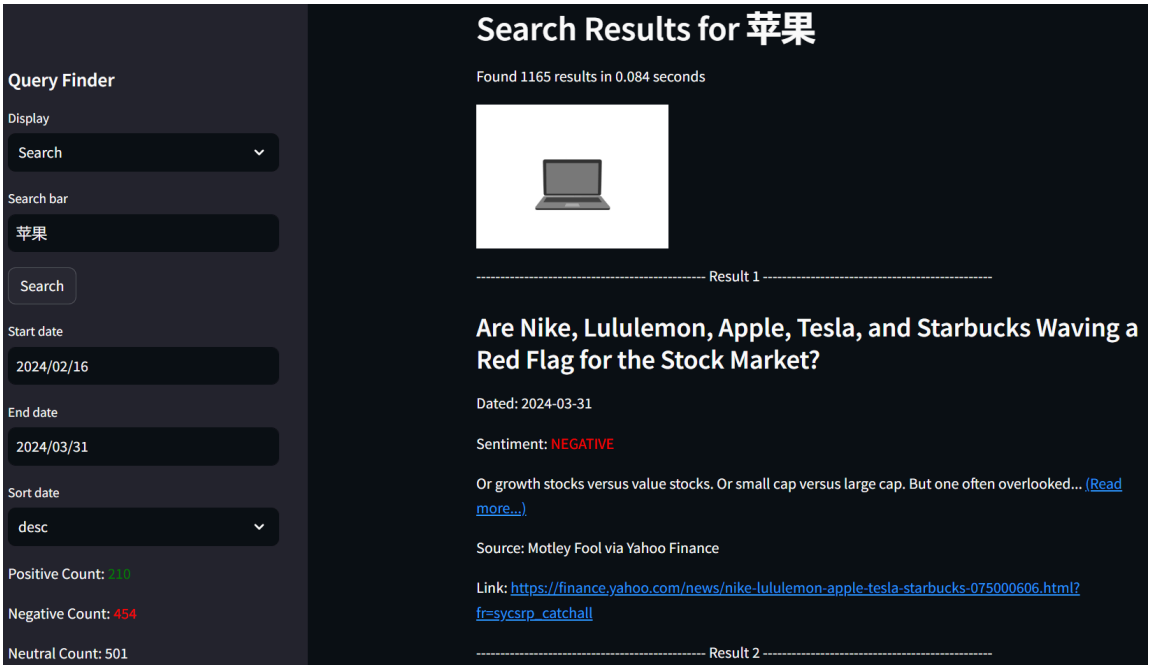

Figure 15: Query in Simplified Chinese

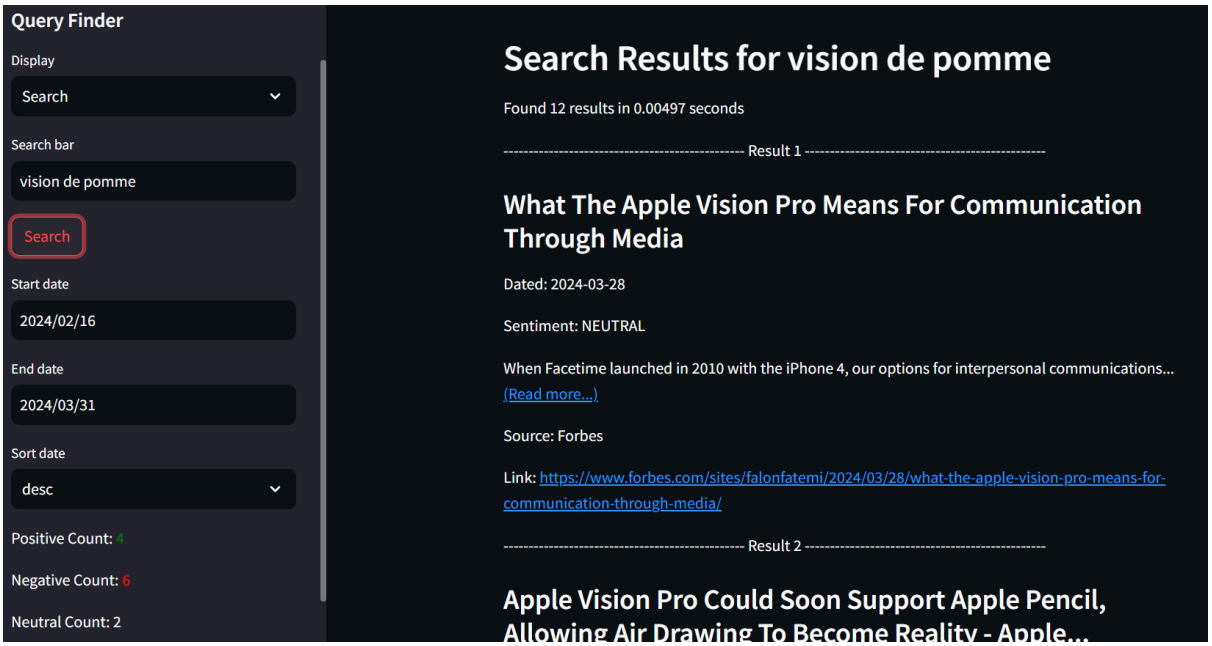

Figure 16: Query in French language

3.4.2 Query suggestion

We also included the functions to provide suggestions if the user inputs an invalid query. The UI will detect if the query has a possible spelling error and display to users with a “Did you mean” with the possible correct spelling of the query.

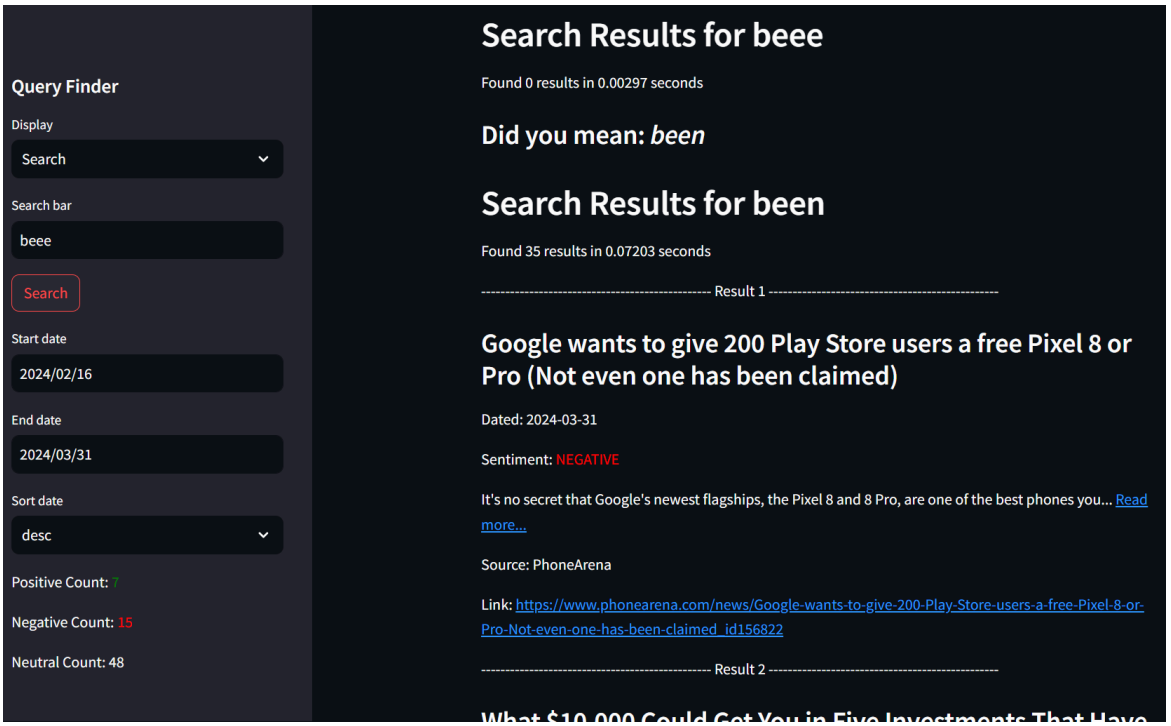

Figure 17: Spelling Checker and Suggestion 1

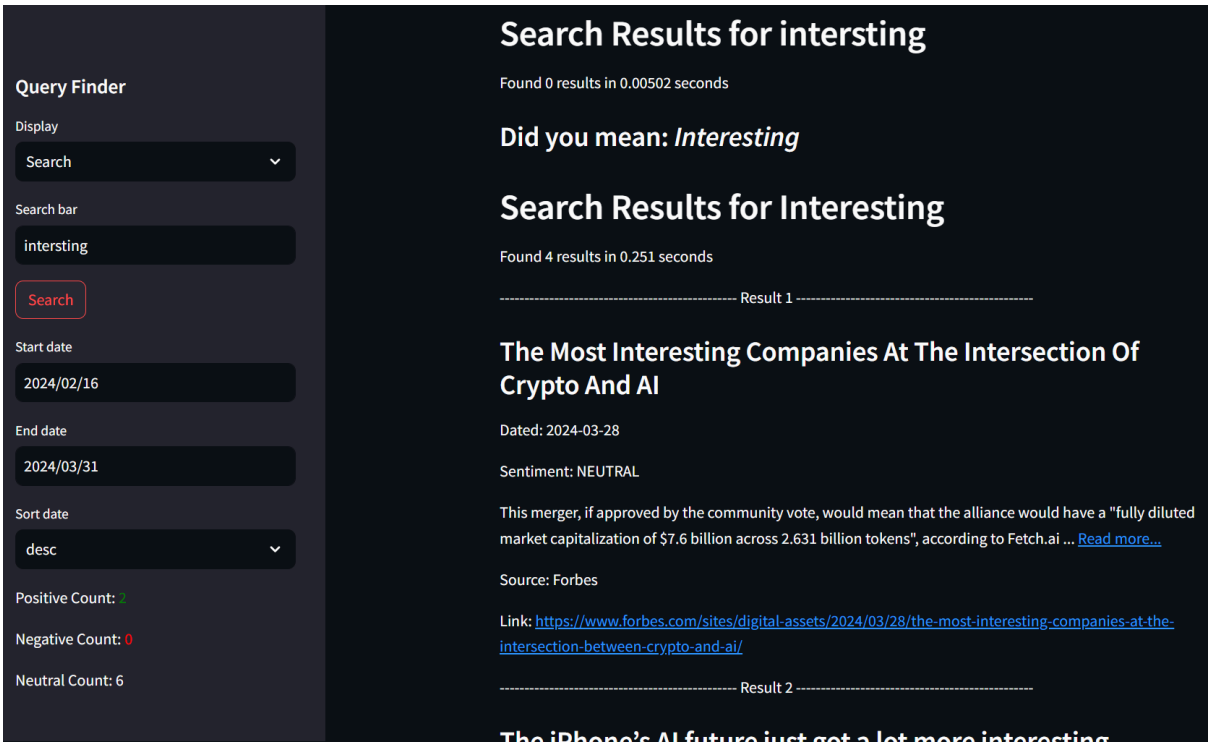

Figure 18: Spelling Checker and Suggestion 2

3.4.3 Sentiment Analysis

This project advocates for the integration of sentiment analysis into the User Interface (UI) to augment user understanding and decision-making based on query results. By incorporating sentiment analysis algorithms, the UI will categorise documents retrieved from user queries as positive, negative, or neutral, displaying corresponding sentiment counts graphically for enhanced interpretability. This integration aims to provide users with deeper insights into the emotional tone and context of retrieved content, facilitating informed decision-making and improving overall user experience. Considerations include algorithm selection for accurate results, scalability for efficient processing of large datasets, and seamless integration into the UI design to optimise usability.

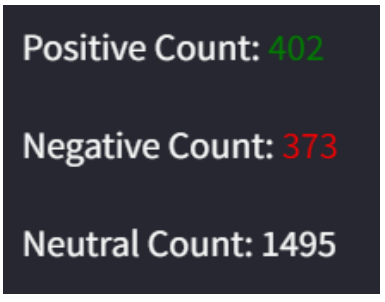

Figure 19: Sentiment Count Feature

3.4.4 Synonym and antonym suggestion

This project proposes the integration of synonym and antonym suggestions into the search functionality to augment user query refinement and search result relevance. By presenting alternative terms with similar or opposite meanings before retrieving search results, the system aims to enhance query precision and assist users in exploring a broader range of relevant content. Synonym suggestions offer users variations of their query to consider, while antonym suggestions enable them to explore contrasting perspectives or concepts. This integration streamlines the search process, empowering users to refine their queries effectively and discover more relevant information with greater ease. Considerations for implementation include robust semantic analysis algorithms, intuitive user interface design, and performance optimization to ensure seamless integration and an optimal user experience.

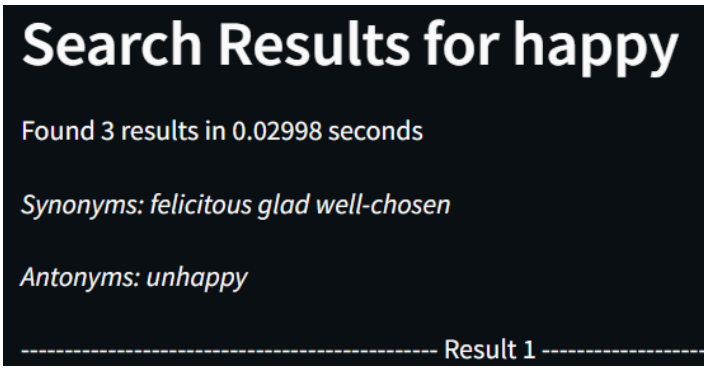

Figure 20: Synonym and antonym suggestion feature

3.4.5 Database Analysis page

The Database Analysis page UI feature offers insightful visualisations to comprehend the underlying data comprehensively. It includes a stacked bar chart illustrating the daily document count by term for the project's main stocks, namely Nvidia, Amazon, Google, Microsoft, and Apple, facilitating a clear understanding of document distribution over time. Additionally, a word cloud visually represents the frequency of words in the database, effectively capturing significant terms while excluding stopwords. Furthermore, a pie chart provides a succinct overview of word frequency distribution for the main stocks, aiding in identifying prominent terms associated with each stock. Lastly, a stacked bar chart delineates stock sentiment analysis by date for the five main stocks, portraying sentiment dynamics categorised as positive, negative, and neutral over time, thereby offering valuable insights into market sentiment trends.

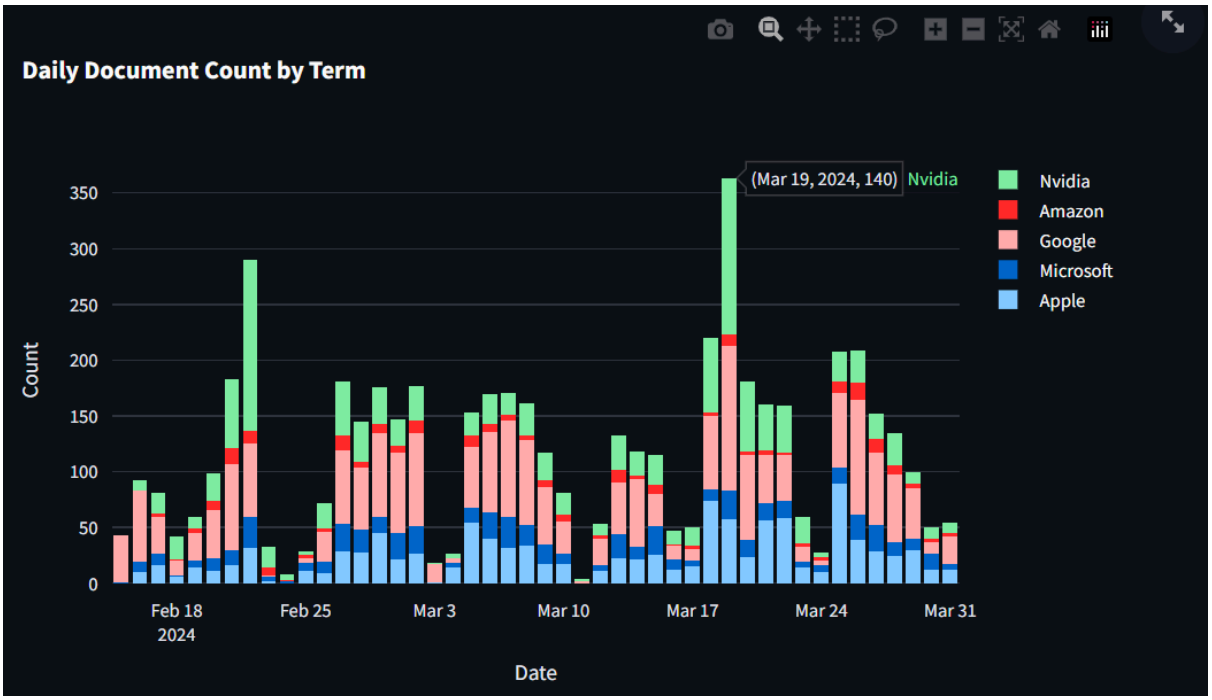

Figure 21: Daily Document Count by Term Stacked Barchart

Figure 22: Wordcloud of Database with stopword removed

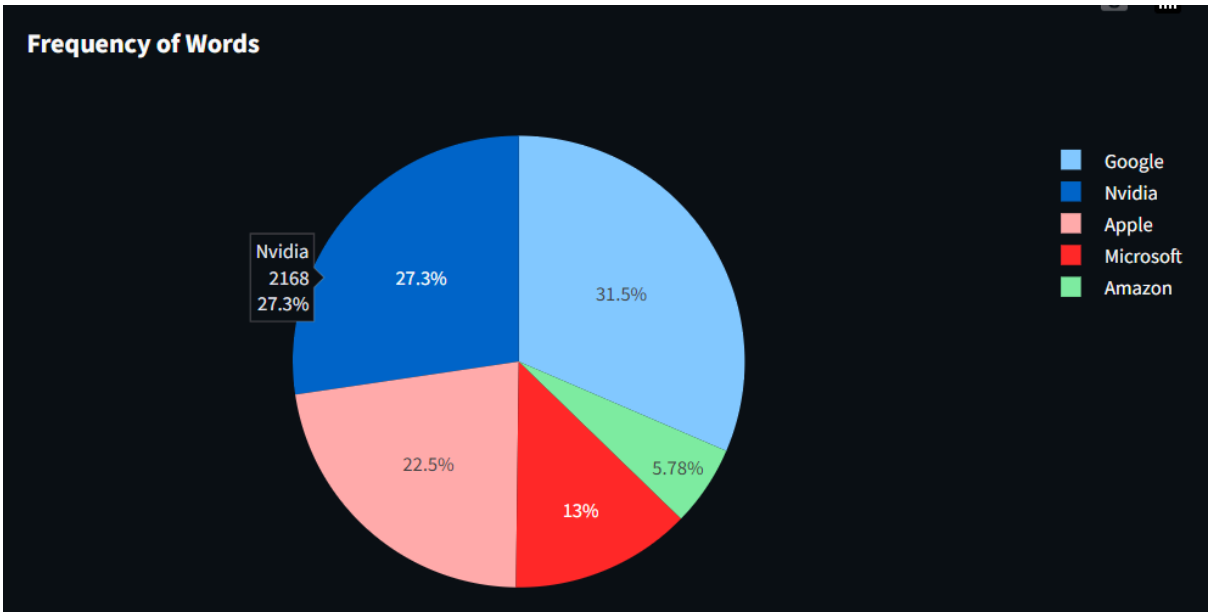

Figure 23: Frequency of Words pie chart

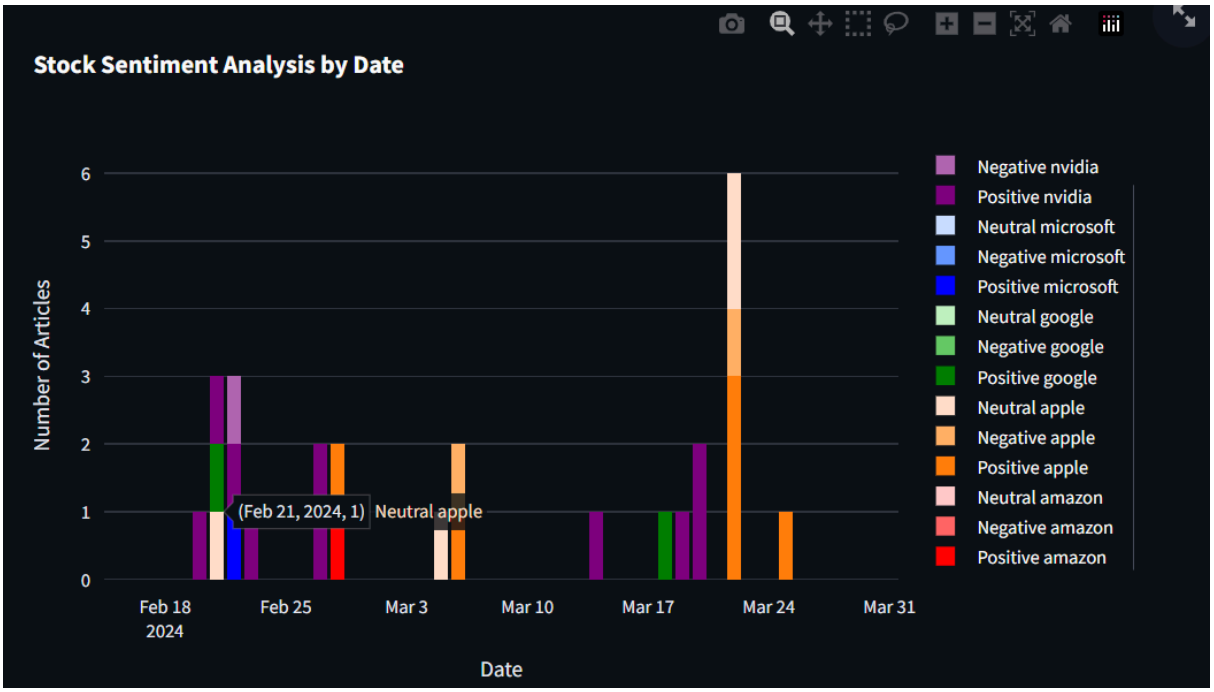

Figure 24: Stock Sentiment Analysis by Date stacked barchart

4. Classification

Sentiment analysis is a complex task with different subtasks such as preprocessing the text before the actual classification of the sentiment. For our project, we will be looking into both

subjectivity and polarity detection to detect neutral versus opinionated and positive versus negative respectively. Both classification approaches will contribute significantly to our project by offering insights into market sentiment and perspectives, which are essential for investors to make informed decisions. Different models will also be explored to find the most suitable model to predict the sentiments. Enhancement methods like the ensemble method will also be discussed in this section.

4.1 Data Preprocessing

4.1.1 Data Cleaning and Preprocessing

The financial stock headlines first need to be preprocessed before performing classification.

We first tokenize the headlines to obtain each word and proceed to perform the relevant cleaning. We then removed any special characters before removing stop words and punctuation using the NLTK library.

Finally lemmatization and stemming is performed so that the root of each word can be obtained to ensure an accurate classifier. In the stock headlines there could be many variations of important words. For example, falling, fallen and fell would be converted to fall so that they all have the same negative connotation.

4.1.2 Data Labelling

To build the evaluation dataset, 2 annotators were chosen to determine the sentiment of the article headlines. There are 3 columns for labelling the sentiment. “Sentiment1” and “Sentiment2" represented the sentiments given by annotators 1 and 2 respectively. The sentiment that is ultimately agreed upon will be listed in the "FINAL Sentiment" column. If sentiment 1 and 2 share the same label, the final sentiment will be identical. In case of disagreements, the two annotators will discuss with each other to agree on one sentiment. Note that there is no data labelled as “Opinionated” since any data labelled as “Positive” and “Negative” is considered as “Opinionated”. The table below summarises the statistics of the labelled data.

Subjectivity Detection		Polarity Detection	
Neutral	Opinionated	Positive	Negative
500	1209	608	601
Total: 1709		Total: 1209	

Table 6: Sentiment Annotation Summary

To calculate the inter-annotator agreement score, we used the Fleiss' Kappa formula on our manually labelled data. The calculation is done using Python code and the inter-annotator agreement score is 96.6% [1].

```
In [8]: from sklearn.metrics import cohen_kappa_score

        def calculate_fleiss_kappa(data):

            n_items = data.shape[0]
            n_raters = data.sum(axis=1)[0]
            n_categories = data.shape[1]

            # Proportion of all assignments to the j-th category
            p_j = data.sum(axis=0) / (n_items * n_raters)

            # extent to which raters agree for the i-th subject
            P_i = ((data**2).sum(axis=1) - n_raters) / (n_raters * (n_raters - 1))
            P_bar = P_i.mean()  # Mean of these proportions

            # expected value of P_i
            P_e = (p_j**2).sum()

            kappa = (P_bar - P_e) / (1 - P_e)

            return kappa

        fleiss_kappa_value = calculate_fleiss_kappa(data)
        print(f"The Fleiss' Kappa value is: {fleiss_kappa_value*100:.1f}%")

        The Fleiss' Kappa value is: 96.6%
```

Figure 25: Code Snippet for Fleiss' Kappa Calculation

4.2 Model Training

4.2.1 TF-IDF

TF-IDF stands for Term Frequency-Inverse Document Frequency. Term frequency is the frequency in which a certain term appears in the document. It can be measured in several ways, such as number of times it appears, term frequency that is adjusted with length of documents, logarithmic scaled frequency, or boolean frequency.

Term frequency	
n (natural)	$\mathrm{tf}_{t,d}$
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$

Figure 26: Term frequency formulas

Inverse document frequency is used to determine the rarity of a word in the documents. The formula for the calculation of IDF can be shown below.

$$\mathrm{IDF}(t) = \log \frac{1+n}{1+\mathrm{df}(t)} + 1$$

Equation 1: IDF Formula

Then, the TF-IDF is calculated by multiplying the TF and the IDF values. TF-IDF measures the importance of a word in the documents. A higher TF-IDF is due to higher TF, implying that a term appears frequently within a document, as well as a higher IDF, implying that a term is rare across all documents in the collection.

4.2.2 N-grams

By default, TF-IDF utilises unigrams for extracting features from text documents. Unigrams focus solely on singular words, potentially overlooking valuable sentiment-related nuances in a text. For example, intensifiers like "very good" or "super bad" are neglected which significantly impact sentiment. Additionally, negation words like "not good" or "not bad" may be disregarded, potentially reversing the sentiment conveyed. This limitation could lead to inaccurate classification results due to the loss of sentiment information. Incorporating bigrams or even trigrams in the feature extraction process has the potential to address this issue by capturing a broader range of sentiment expressions. To evaluate the impact of adding bigrams or trigrams as features, we experimented with three document-feature matrices: one consisting of unigrams alone, one incorporating both unigrams and bigrams, and another one incorporating unigrams, bigrams and trigrams.

4.2.3 BERT word embeddings

BERT (Bidirectional Encoder Representations from Transformers) is a powerful natural language processing (NLP) model developed by Google. It generates contextualised word representations that are used as embeddings for downstream NLP tasks. Contextualised word representations are generated based on the entire context of a word within a sentence, enabling capture of the nuances of word meaning and usage in different contexts. Given the Transformer architecture of BERT, it has self attention mechanisms that allow long range dependencies and semantic relationships between words in a sentence to be captured. On top of this, BERT is pre-trained on large text corpora using two unsupervised learning tasks of Masked Language Modelling (MLM) and Next Sentence Prediction (NSP).

Masked Language Modelling (MLM): involves predicting masked tokens within the input sequence, where 15% of the WordPiece tokens are randomly masked. The final hidden vectors corresponding to these masked tokens are used to predict the masked words through an output softmax over the vocabulary. To address potential mismatches between pre-training and fine-tuning, the masking strategy replaces masked words with the actual [MASK] token 80% of the time, a random token 10% of the time, and the unchanged token 10% of the time. This approach contributes significantly to the model's understanding of contextual relationships.

Next Sentence Prediction (NSP): This task entails predicting whether a given sentence is the logical continuation of the preceding sentence. When selecting sentences A and B for each pretraining example, 50% of the time, sentence B is the actual next sentence following A (labelled as IsNext), while 50% of the time, it's a random sentence from the corpus (labelled as NotNext). Despite its simplicity, the NSP task offers valuable insights and prepares the model for a range of downstream applications.

BERT can then be fine tuned on specific NLP tasks like sentiment analysis, where the model can be adapted to specific downstream tasks by merely introducing an additional output layer. Using the contextualised word embeddings generated by BERT, we can better understand how different words relate to one another in 'Headlines' and determine the sentiment of the news article from there.

However BERT, like many other pretrained language models may contain lookahead bias, since it has knowledge of the impact and outcomes of historical events.

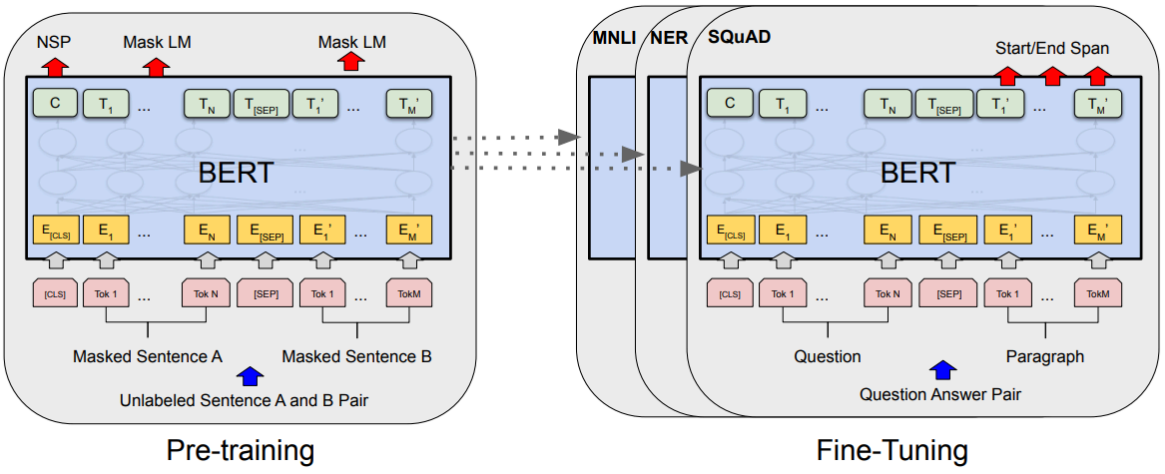

Figure 27: Bert Pre-training and Fine-Tuning Process

4.2.4 Global Vectors for Word Representation (GloVe)

GloVe is a popular word embedding technique, developed as a way to efficiently capture the linear substructures of words in a vector space. GloVe embeddings are derived from the statistical information about word occurrences in a corpus of text, specifically through the

co-occurrence probabilities of words. Unlike TF-IDF which produces sparse vectors focused on term uniqueness across documents, GloVe is focused on the semantic meanings in a dense vector space, capturing both global statistics and local context. The following are the key steps in generating GloVe embeddings.

Co-occurrence Matrix Construction: The first step in GloVe is constructing a large matrix of co-occurrence information. This matrix counts how frequently each word occurs in the context of every other word within a given corpus. The context is usually defined by a window of neighbouring words. For example, in a window size of 10, the model counts how often each word appears within 10 words of every other word.

Matrix Factorization: The co-occurrence matrix is then transformed by focusing on the ratios of word-word co-occurrence probabilities rather than the probabilities themselves. The idea is that these ratios can capture more meaningful information such as semantic and syntactic relationships between words. The GloVe model applies matrix factorization techniques to approximate these ratios, effectively reducing the dimensionality of the matrix while preserving relational information.

Learning Word Vectors: The actual learning of word vectors (embeddings) is done by minimising a loss function that measures the difference between the dot product of the word vectors and the logarithm of their co-occurrence probabilities. The optimization is performed over all word pairs, adjusting the word vectors to better fit the observed patterns in the data.

Vector Representation: The result of this training process is a set of vectors, one for each word in the vocabulary, where distances and directions between vectors correspond to semantic and syntactic relationships. Words with similar meanings will have vectors close to each other in the vector space, whereas words with different meanings will be farther apart.

4.3 Classification Models

4.3.1 Motivation for classification models

Our group chose Naive Bayes, KNN, SVM, and logistic regression with BERT word embeddings because of their proven effectiveness in text classification tasks. Naive Bayes is simple and scalable, making it a pragmatic choice for baseline classification. KNN excels in capturing local patterns in data, while SVM provides robustness against outliers and high-dimensional feature spaces. Logistic regression with BERT word embeddings makes use of contextual embeddings to identify semantic nuances within the text to improve classification accuracy.

In addition, ensemble learning techniques such as voting and stacking were used to capitalise on the diversity of the individual classifiers. By combining the predictions of various base models, ensemble learning enhances the overall performance, and helps to mitigate the risk of overfitting and improves classification robustness.

The classification models also extended into cognitive learning methods like echo state networks, which offer unique capabilities in handling temporal dependencies and nonlinear dynamics. This innovative approach allows the model to adapt dynamically to evolving patterns in sentiment, enriching the classification process with insights derived from temporal context.

The SOTA model for sentiment classification on the 20NEWS dataset is TF-IDF, with SVM. The top few SOTA on IMDB movie reviews dataset with the binary sentiment polarity labels were pre trained language models which may contain lookahead bias in the context of backtesting trading strategies. For example if the pre-trained language models were trained on pre-2013 data and in 2012 there are many news articles on how company A's new phone is a flop and over hyped but in 2011 the articles were hyping the new phone. When performing a backtest, the model at 2011, with pretrained data past 2011 might have a different sentiment compared to if it was only pretrained with pre-2011 data. Using only the embeddings reduces the look-ahead bias since it is designed to be temporally neutral. The embeddings capture on a generalised representation that captures various dimensions of a word's usage rather than

specific event outcomes or sentiments associated with that word at different times. This captures the broader use across contexts instead of being tied to specific temporal events. Thus the resulting word embeddings are robust and versatile, able to function effectively across different temporal contexts without inheriting future-biased sentiments.

In the paper An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation, an accuracy score of 89.91% was achieved on the IMDB dataset, causing it to be competitive with neural network based models.

30	BP-Transformer + GloVe	92.12	×	BP-Transformer: Modelling Long-Range Context via Binary Partitioning		2019	Transformer
31	BCN+Char+CoVe	91.8	×	Learned in Translation: Contextualized Word Vectors		2017	
32	ToWE-SG	90.8	×	Task-oriented Word Embedding for Text Classification		2018	
33	LSTM with dynamic skip	90.1	×	Long Short-Term Memory with Dynamic Skip Connections		2018	LSTM
34	CNN+LSTM	88.9	×	On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis		2017	CNN LSTM
35	UnICORNN	88.4	×	UnICORNN: A recurrent model for learning very long time dependencies		2021	
36	CfC	88.4	×	Closed-form Continuous-time Neural Models		2021	

Figure 28: Accuracy Comparison of Text Sentiment Classification Models on IMDB Dataset

Although 20NEWS task is to classify news groups, the tasks are fundamentally about categorising text into predefined categories. Furthermore due to the poor quality of publicly labelled datasets, the models need to be able to withstand a high amount of noise. Also this technique is domain independent which makes them applicable to different types of text data, irrespective to context.

TF-IDF combined with Support Vector Machines (SVM) is effective for tasks characterised by clear lexical distinctions and stable data distributions. However, this approach may be less effective in processing nuanced language uses, such as irony or sarcasm, or when managing very large vocabularies without the application of dimensionality reduction techniques. Nonetheless, financial headlines typically exhibit a more serious tone with minimal use of irony or sarcasm.

The paper Neuromorphic Sentiment Analysis Using Spiking Neural Networks, claims a 100% accuracy on the IMDB dataset. This scores the paper higher than current SOTA models and hence motivates the use of spiking neural networks and its variants in this project.

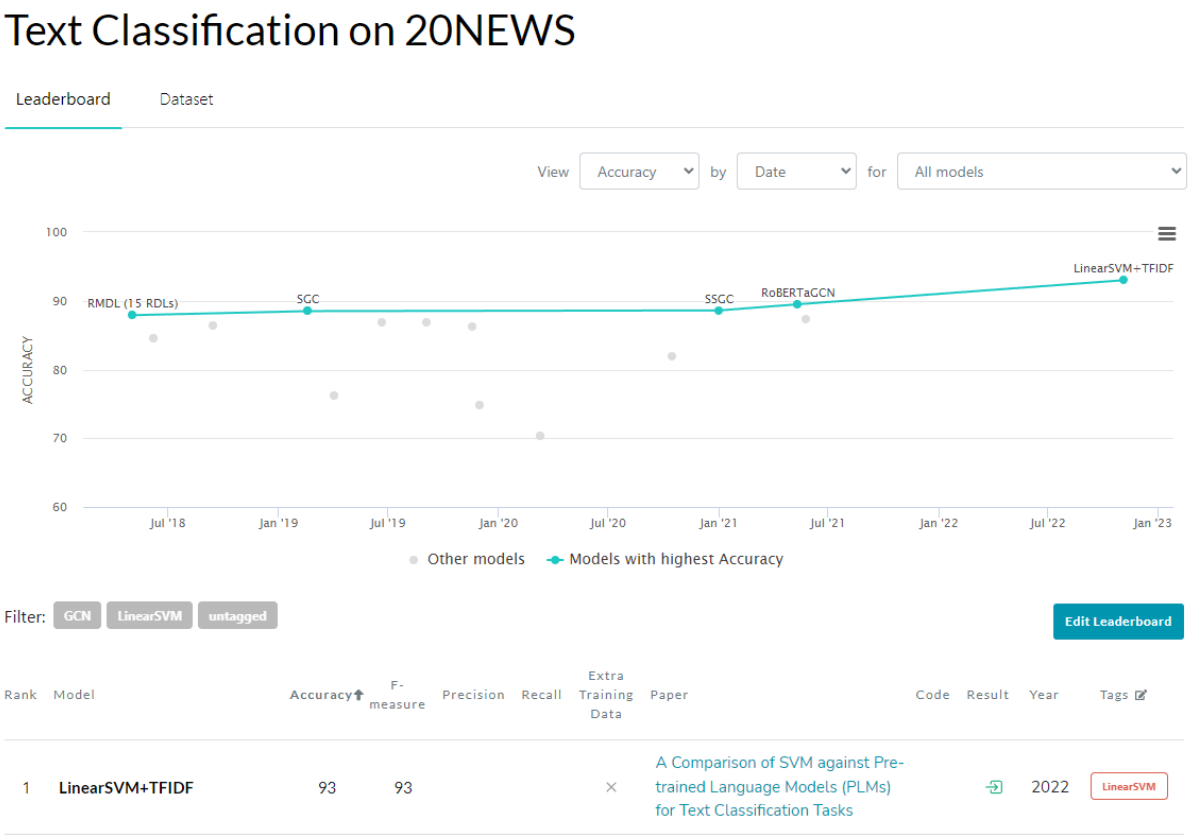

Figure 29: Accuracy comparison of Text Classification on 20NEWS

4.3.2. Models Used

4.3.2.1 Support Vector Machine (SVM)

The goal of SVM is to find the optimal hyperplane in an N-dimensional space that classifies the data points into different classes. A hyperplane is a decision boundary that separates different classes. The optimal hyperplane is the one that maximises the margin between the two classes. Margin is the distance between the hyperplane and the nearest data points from either class. Support vectors are data points that are closest to the hyperplane and influence its position and orientation.

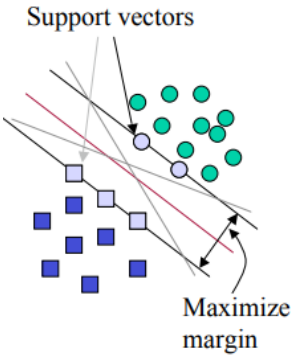

Figure 30: Hyperplane and Support Vectors in SVM

4.3.2.2 Naive Bayes

Naive Bayes is a classification algorithm based on Bayes' theorem, whose formula is as follows:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Equation 2: Conditional Probability Formula

where P(A) is the prior probability of class A, P(B) is the prior probability of predictor B,
P(A | B) is the posterior probability of class A given predictor B,
P(B | A) is the likelihood, the probability of predictor B given class A.

The assumption of Naive Bayes is that all features are independent of each other. Bayes' theorem is used to predict the probability that a given feature set belongs to a particular class. The class that has the highest probability is the output prediction.

4.3.2.3 Logistic Regression (LR)

Logistic Regression is a statistical and machine learning technique used for binary or multi-class classification tasks. The output is calculated using the sigmoid function to ensure that the output probability lies between 0 and 1. The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

Equation 3: Logistic Regression Formula

where $z = w \cdot x + b$ is the linear combination of the input features x with the weight vector w and bias b.

A decision boundary needs to be defined for logistic regression. Points on one side of the boundary are classified as class 0, and points on the other side as class 1.

4.3.2.4 Random Forest

A Random Forest is composed of many individual decision trees that operate as an ensemble. Each tree in the Random Forest makes an independent decision, and the final output is determined by aggregating the results from all the trees. During the construction of each decision tree, randomness is introduced by selecting a subset of features randomly at each node for splitting, rather than considering all features. An example random forest decision tree is as follows:

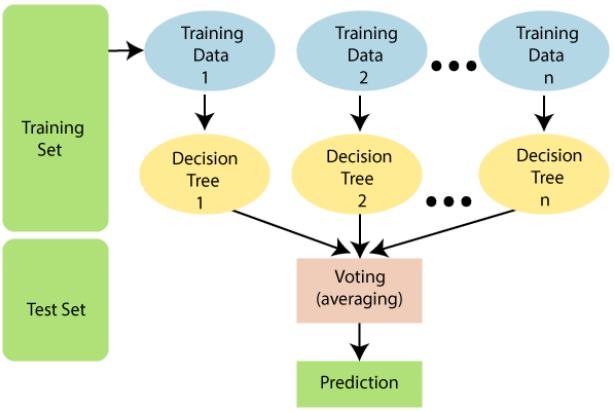

Figure 31: Random Forest illustration

4.3.2.5 K Nearest Neighbour (KNN)

KNN operates on the principle of proximity or similarity. The idea is that data points with similar features tend to belong to the same class. When a new data point needs to be classified, KNN looks at the class labels of its K nearest neighbours in the feature space and assigns the majority class label among those neighbours to the new data point. Usually, the value of K is chosen as an odd number to avoid ambiguity in the model.

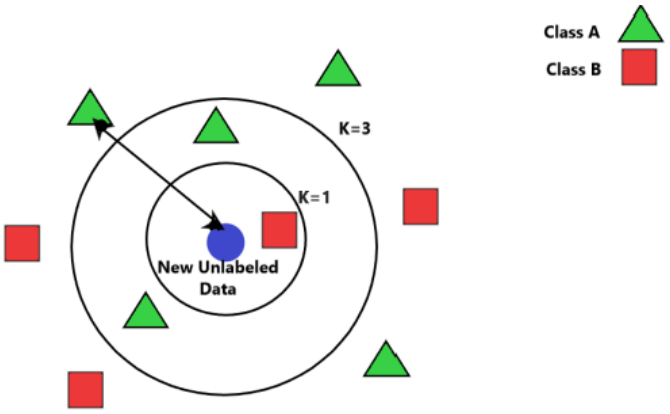

Figure 32: KNN illustration

4.3.2.6 SGD Classifier

The Stochastic Gradient Descent (SGD) Classifier with log loss uses stochastic gradient descent for training. Gradient descent is a first-order iterative optimization algorithm used for finding the minimum of a function. This project uses it to minimise the logistic loss function

$$L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\sigma(x_i^T\theta)) + (1-y_i)\log(1-\sigma(x_i^T\theta))\right]$$

Equation 4: Logistic Loss Function

L(θ), as shown above, which measures the discrepancy between the observed outcomes and the probabilities predicted by the model. To minimise the logistic loss function using gradient descent, the gradient is calculated. The gradient of the logistic loss function with respect to θ is:

$$\nabla_\theta L(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i - \sigma(x_i^T\theta))x_i$$

Equation 5: Gradient of Logistic Loss Function

The parameter θ are updated iteratively in the direction that reduces the logistic loss function,

$$\theta_{\text{new}} = \theta - \alpha\nabla_\theta L(\theta)$$

Equation 6: Update of Parameters

where α is the learning rate, a positive scalar that determines the step size of the update. The optimization process is repeated until convergence, which occurs when the change in the loss function is below a predefined threshold, or a maximum number of iterations is reached.

SGD improves efficiency compared to GD by using only a small batch of samples to perform each iteration of the optimization process, reducing the memory requirements as the model's parameters are iteratively updated using a single training example. This is especially useful when handling large datasets with and their larger vector representations.

4.3.2.7 Long Short-Term Memory (LSTM)

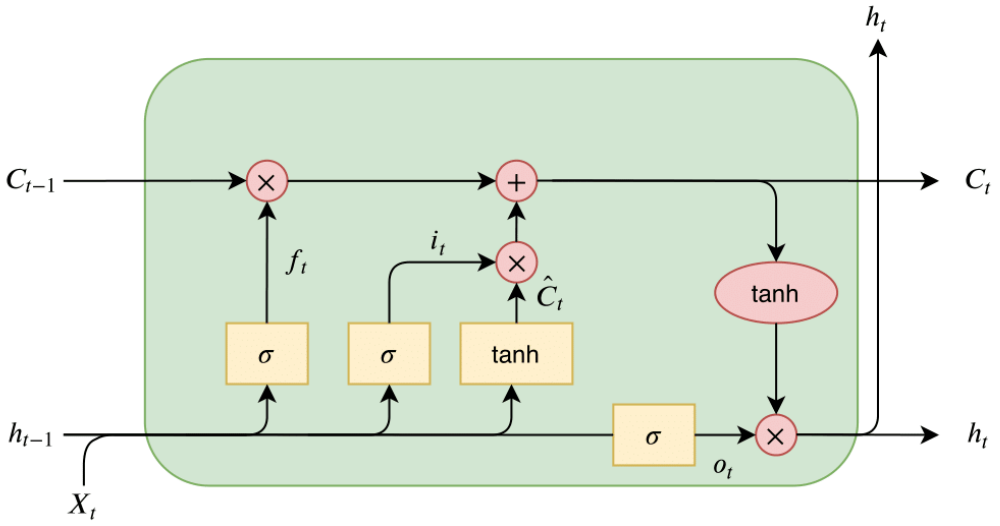

Figure 33: LSTM Illustration

LSTM is an advanced type of Recurrent Neural Network (RNN) designed to address the issue of long-term dependencies in sequence data. An LSTM unit comprises several gates that control the flow of information. These gates are the forget gate (f_t), input gate (i_t), and output gate (o_t), each of which performs a specific function. The cell state (C_t) acts as a conveyor belt, carrying relevant information throughout the sequence of data.

Forget Gate (f_t): This gate decides which information should be discarded from the cell state. It looks at the previous hidden state (h_{t-1}) and the current input (x_t), and passes them through a sigmoid function (σ), which outputs a number between 0 and 1. A value of 0 means “completely forget this” while a value of 1 means “completely retain this.” The result is multiplied element-wise (⊗) to the previous cell state (C_{t-1}).

Input Gate (i_t) and Candidate Values (C̃_t): The input gate decides which new information will be stored in the cell state. Simultaneously, a tanh layer creates a vector of new candidate

values, $\tilde{C}_t$, which could be added to the state. Both i_t and $\tilde{C}_t$ are also dependent on h_{t-1} and x_t, and undergo element-wise multiplication, which decides the update to the cell state.

Cell State Update (C_t): The old cell state (C_{t-1}) is updated to the new cell state (C_t). The old state is multiplied by the output of the forget gate and then adds the element-wise product of the input gate and candidate values ($i_t \otimes \tilde{C}_t$). This is the key step that allows LSTMs to carry relevant information through the sequence of data, potentially over long time intervals.

Output Gate (o_t) and Hidden State (h_t): Finally, the output gate decides what the next hidden state should be. The hidden state contains information on previous inputs. The cell state is passed through a tanh (pushing the values to be between -1 and 1) and multiplied by the output of the sigmoid gate on h_{t-1} and x_t, so only the parts of the cell state determined by the output gate are outputted.

By using these gates and the cell state, LSTMs selectively remember patterns over long time intervals and ignore irrelevant data, making them highly effective for tasks such as time-series prediction, language modelling, and other sequence-related tasks.

4.3.2.8 Convolution Neural Networks (CNN)

CNN is a type of deep neural network that is powerful for processing data with grid-like topology such as images. However, CNN could also be used to perform natural language processing tasks by considering text sequence as one-dimensional image. 1D CNNs are very effective for extracting local features, such as n-grams. The core components of CNN include:

1. Convolutional layers: Apply filters on inputs to produce feature maps. Each filter detects specific features.
2. Activation functions: Introduce non-linearity into the model to discover complex patterns in the data.
3. Pooling layers: Reduce spatial dimension of the input, making the detection of features more robust.

4. Fully connected layers: Interpret the extracted features and perform classification.
5. Output layer: usually uses a softmax function to output probability distribution over the classes.

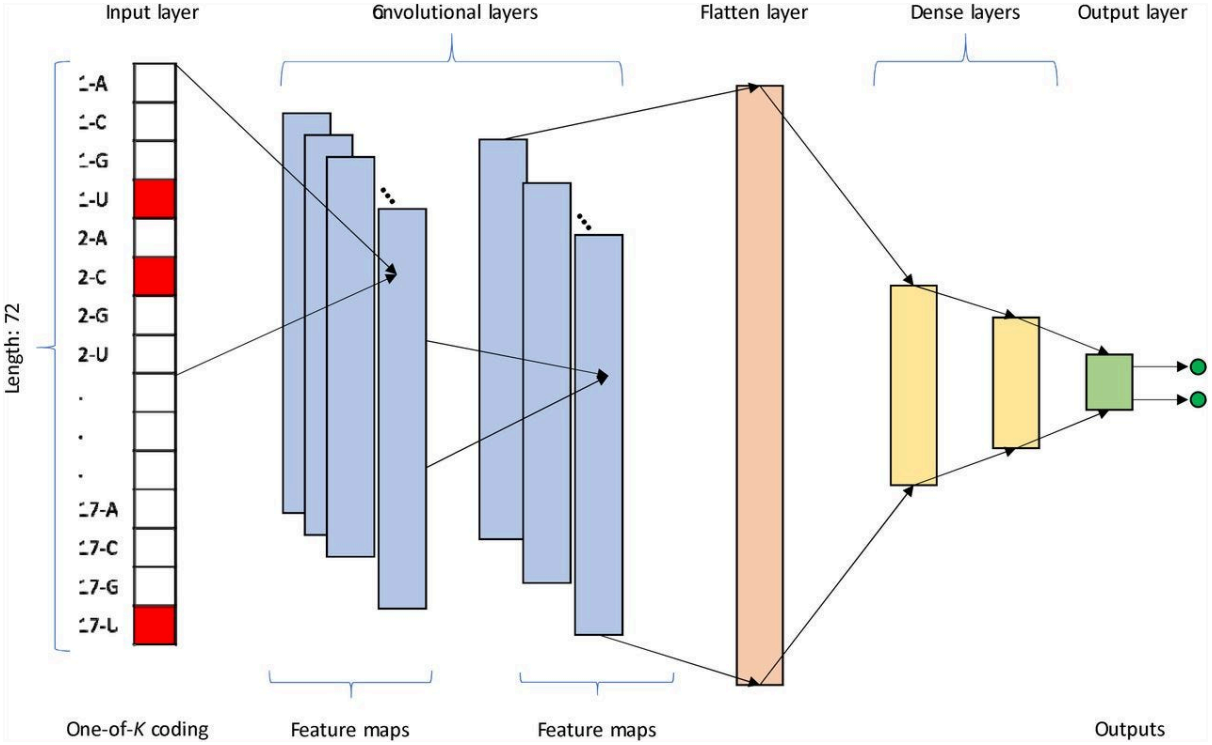

Figure 34: CNN Architecture

4.3.2.9 Brain Inspired Algorithms

4.3.2.9.1. Echo State Networks

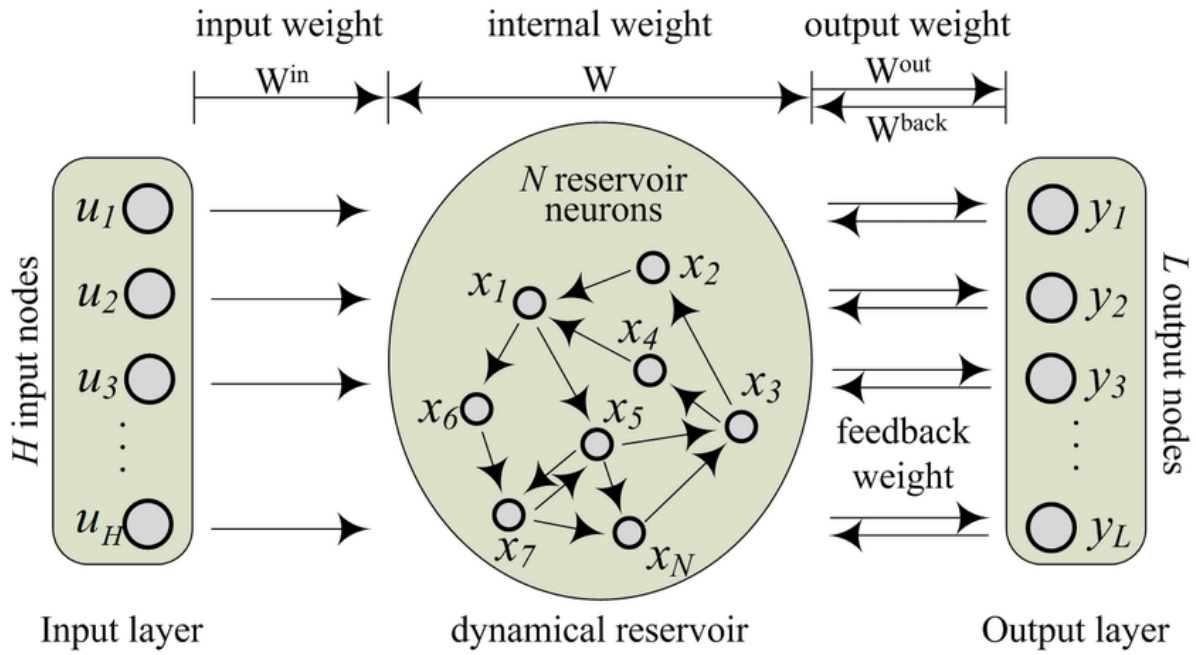

Figure 35: Echo State Networks Architecture

An Echo State Network (ESN) is a type of recurrent neural network (RNN) that belongs to the larger class of reservoir computing methods.

1. Reservoir:
The core of an ESN is the reservoir, which is a large, fixed, sparsely connected network of neurons. Unlike traditional RNNs, the weights within the reservoir (internal connections) are randomly initialised and remain unchanged during training. This reservoir acts as a dynamic memory that transforms and stores the history of input sequences in its state.

2. Echo State Property:
The defining characteristic of ESNs is the "echo state property," which ensures that the state of the reservoir depends on the current input and the fading memory of past inputs. This property guarantees that the reservoir's state is a function of past inputs, but older inputs have progressively less influence.

3. Sparse Connectivity:
In the reservoir, neurons are sparsely connected. This sparsity contributes to the efficiency and dynamic richness of the network, allowing it to model complex temporal patterns with fewer resources than a fully connected network.

4. Training:
Training an ESN is significantly simpler and faster than traditional RNNs because only the weights connecting the reservoir to the output (readout weights) are adjusted. The input-to-reservoir and reservoir-to-reservoir weights remain fixed, eliminating the need for complex backpropagation through time or gradient descent methods, which are computationally intensive and prone to problems like vanishing or exploding gradients.

Similar to a brain, where not every neuron is connected to every other neuron. There's a level of sparse connectivity, much like the reservoir in an ESN, where each neuron is only connected to a small subset of other neurons.
In an ESN, the internal connections within the reservoir are randomly initialised and remain largely unchanged. This mirrors the way certain synaptic connections in the brain are long-lasting and change relatively slowly compared to the rapid neural dynamics associated with processing immediate sensory inputs or performing quick computations. This aspect of ESNs aligns with the understanding that while the brain's structure provides a stable framework, the dynamics within that structure are what enable complex processing.

4.3.2.9.2 Spiking Neural Networks

Spiking neural networks (SNNs) are a type of artificial neural network that is inspired by the characteristics of biological neural networks. They aim to mimic the way neurons in the brain operate, which includes the use of spikes for information transmission and processing. Unlike traditional artificial neural networks, which process information in a continuous manner, SNNs work on the basis of discrete events in time, called spikes.

Rate encoding is a fundamental mechanism used by biological neurons to represent information. The frequency of action potentials or spikes can encode the intensity of a stimulus or the strength of a signal. In spiking neural networks, this principle can be used to

process and transmit information efficiently. Different patterns of spikes can represent different sensory inputs, motor commands, or cognitive states. By converting analog signals into spike rates that can be processed in a manner analogous to the way biological neural networks operate, the firing rate of each neuron can be tuned to respond to specific patterns of input, allowing the network to learn and make decisions based on the encoded information.

Rate encoding faces challenges due to the relatively low firing rates of individual neurons. This slow rate seems incompatible with the rapid reaction times observed in animals and humans. Population coding addresses this by considering the collective output of groups of neurons rather than individual neuron responses. In this model, the firing rates of multiple neurons are pooled together. This method allows the brain to represent and process information much more rapidly than would be possible if relying solely on the rate of firing of individual neurons. By aggregating the spikes from a group of neurons within a short time window, it is possible to achieve a higher effective firing rate, enhancing the system's ability to respond quickly to stimuli. The advantage of population coding lies not only in speed but also in its robustness and accuracy in representing complex stimuli. Each neuron in a population might respond to slightly different aspects of a stimulus, and together they can provide a richer, more detailed representation of the sensory input or cognitive state. This diversity among the responses within a neuron population leads to a more stable and reliable encoding of information, as the noise or variability in individual neuron responses is averaged out across the population.

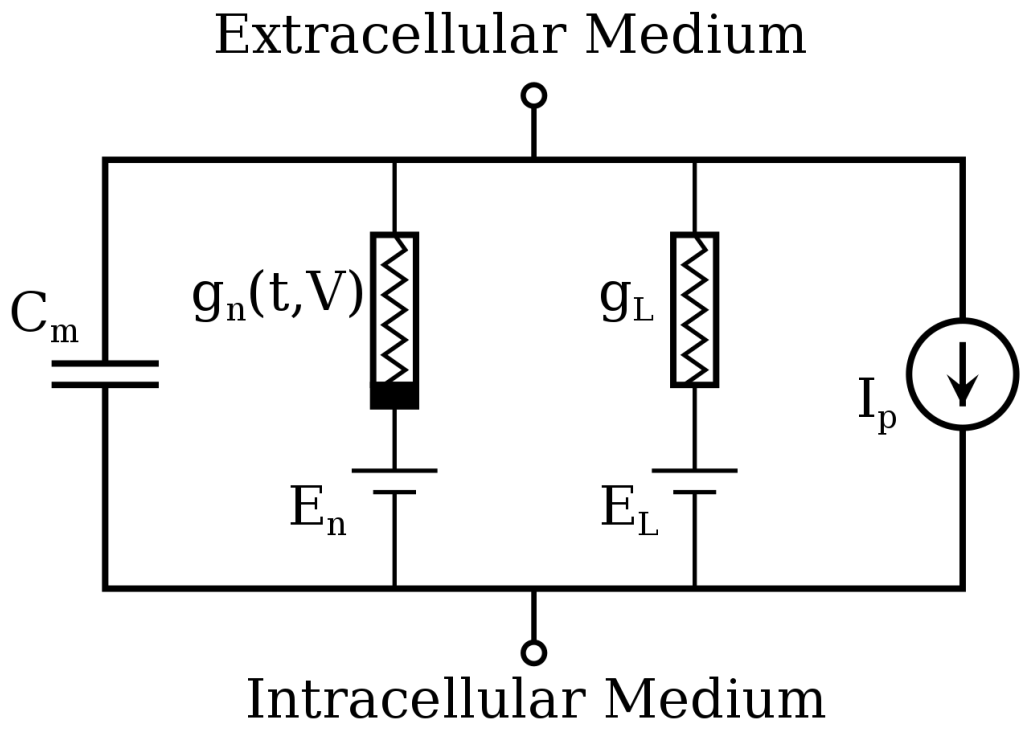

Figure 36: Biological Neuron Membrane Circuit Model

Hodgkin-Huxley Neuron Model most closely describes the ionic mechanisms of a neuron's membrane. It's based on the voltage across a neuron's membrane and the ionic currents that pass through it. The model is composed of a set of four nonlinear differential equations, which are initiated and propagated along the neuron's axon. These equations describe the flow of sodium (Na+) and potassium (K+) ions through their respective channels, as well as the leakage current and the change in membrane potential over time. Solving these equations requires numerical methods that can handle their complexity and nonlinearity, which is computationally intensive. The model is sensitive to initial conditions, requiring fine-tuning of the parameters for accurate simulation, increasing its computational demand. When simulating networks of neurons, each neuron may require its own set of Hodgkin-Huxley equations with distinct parameters, further increasing its computational load.

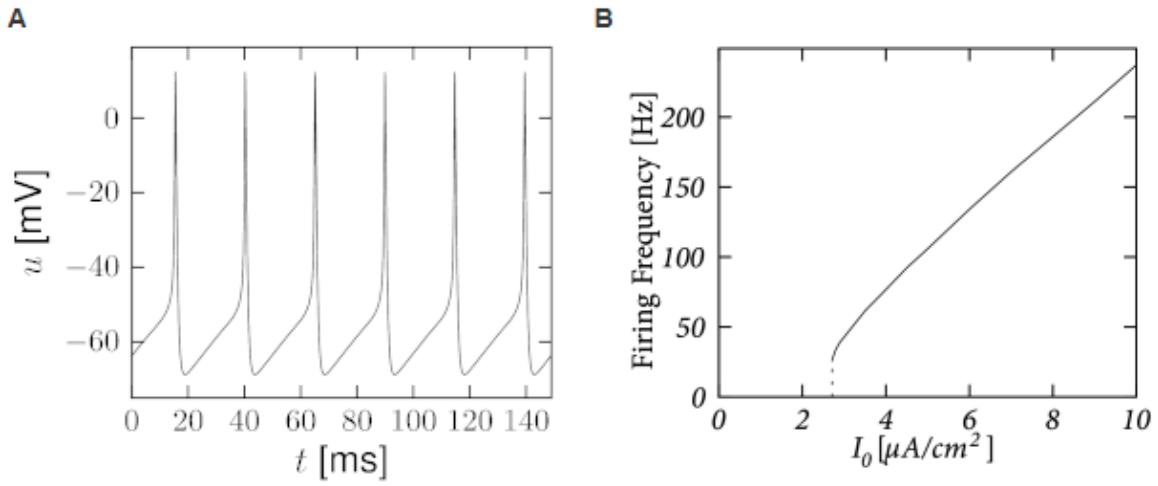

Figure 37: Comparison of Neuronal Spiking Dynamics

When activated, the neuron output signals as spikes, which is depicted in panel A of the figure. These spikes are representative of action potentials, the primary means of electrical communication in neurons. The action potentials are shown as rapid up-and-down deflections in membrane potential (measured in millivolts, mV) over time (measured in milliseconds, ms). The regular pattern of spikes indicates that the neuron is firing repeatedly at a steady rate.

Panel B shows the relationship between the input current (I_0, measured in microamperes per square centimetre, μA/cm²) and the neuron's firing frequency (measured in Hertz, Hz). This is a plot of a neuron's firing rate as a function of injected current, which demonstrates the principle of rate encoding in spiking neural networks.

To avoid high computational cost, this project compromises and utilises a simplified model. The Leaky Integrate-and-Fire Neuron (LIF), takes the sum of weighted inputs, much like the artificial neuron. But rather than passing it directly to an activation function, it will integrate the incoming spikes as a change in membrane potential, over time with a leakage, much like an resistor–capacitor circuit. When the membrane potential exceeds a certain threshold, the neuron generates a spike (a 'fire') and the potential is reset. The 'leaky' part of the name comes from the fact that the model incorporates a form of decay that simulates the gradual loss of membrane potential over time, akin to a leak in a capacitor. The LIF neuron abstracts away

the shape and profile of the output spike; it is simply treated as a discrete event. As a result, information is not stored within the spike, but rather the timing (or frequency) of spikes. This can be seen in the diagrams below, where the shape of the spikes differs from the Hodgkin-Huxley Neuron Model, but still retains the essence.

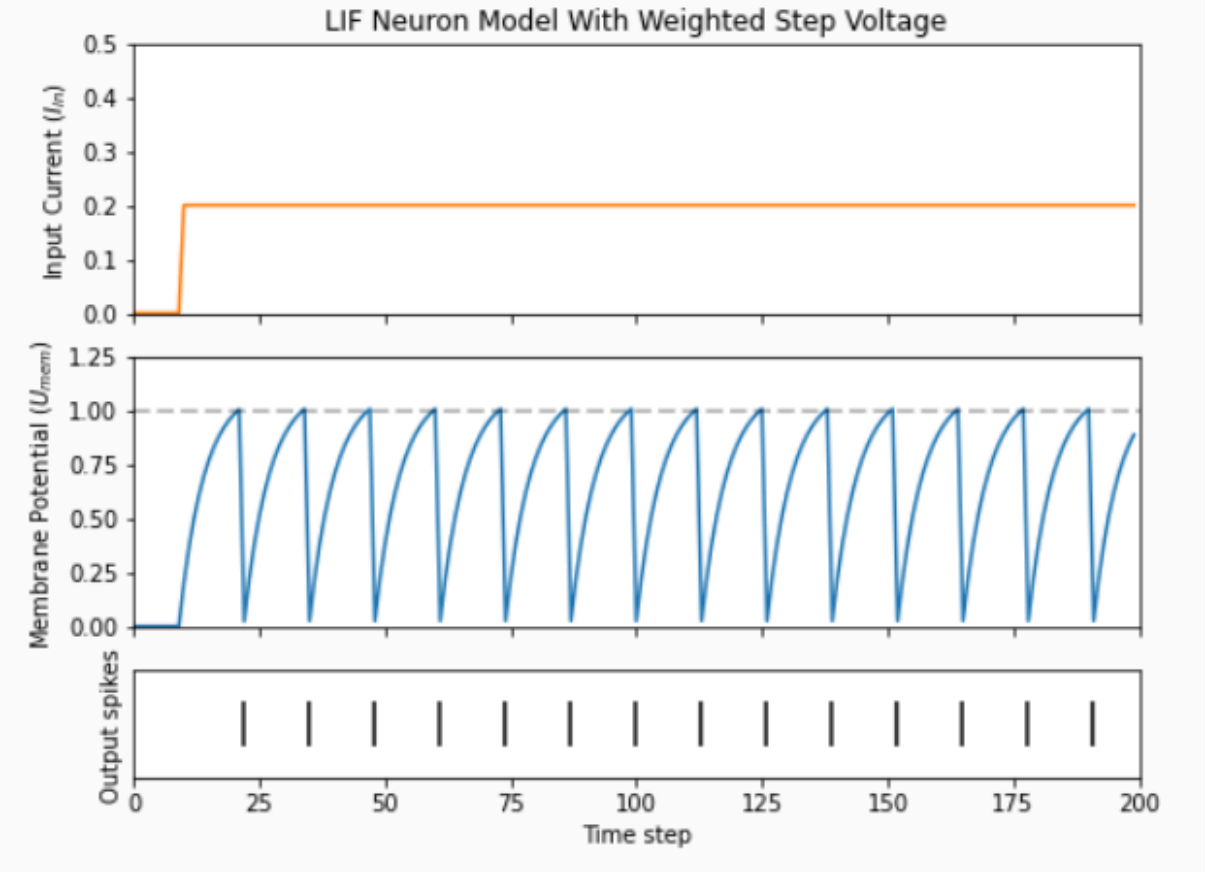

Figure 38: Plot of LIF Neuron Model with Weighted Step Voltage

However due to the non-continuous nature of spikes, which can be represented as a Heaviside step function, The derivative of the Heaviside step function, Dirac Delta function, evaluates to 0 everywhere, except at the threshold where it tends to infinity. This means the gradient will almost always be nulled to zero (or saturated if it sits precisely at the threshold), and no learning can take place, leading to dead neuron problems. To overcome this the Heaviside step function is approximated by a sigmoid function which is shifted to centre on the threshold, this is known as surrogate gradient approach. The chart below shows the backpropagation of the SNN.

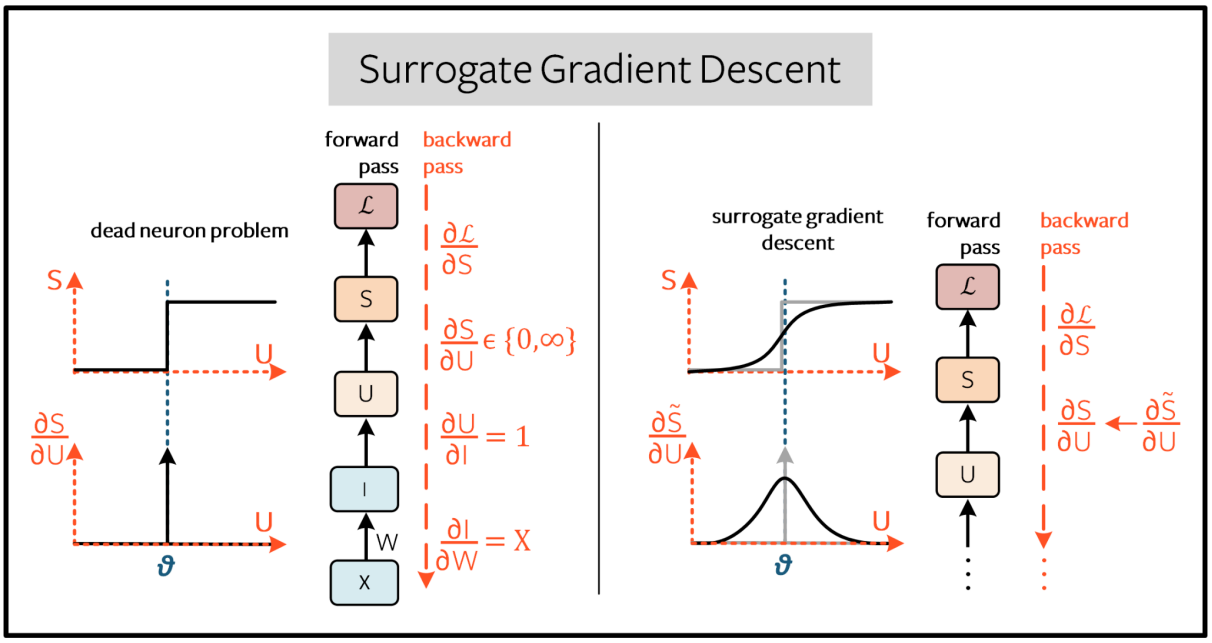

Figure 39: Surrogate Gradient Approach for Spiking Neural Networks

4.3.2.9.3 Neural Circuit Policies With Liquid Time-Constant Networks

Liquid Time-Constant Networks (LTCs) are a new class of recurrent neural network (RNN) models designed for modelling time-series data using time-continuous dynamics governed by ordinary differential equations (ODEs). These models operate differently from conventional RNNs by incorporating liquid time-constants that vary based on the input, allowing the network to adapt its processing speed and coupling sensitivity dynamically.

In the brain, the response characteristics of neurons can vary depending on the context, such as during learning or in response to different types of environmental stimuli. Similarly, in LTCs, the "liquid" time constants allow the network to dynamically adjust the response speed and sensitivity of its neurons based on the input signals. This mimics the adaptive nature of biological neurons that can modify their behaviour based on the synaptic inputs they receive and their internal states.

The brain operates in continuous time, processing signals that continuously vary over time. Traditional artificial neural networks, including most types of recurrent neural networks (RNNs), typically process information in discrete time steps (e.g., frame by frame in a video or step by step in a time series). In contrast, LTCs, much like the brain, are designed to

operate in continuous time using differential equations to model the changing states of neurons. This continuous-time processing is more akin to how biological neurons integrate and fire over time, leading to a more naturalistic simulation of temporal dynamics.

Neural Circuit Policies (NCPs) are a class of neural network architectures inspired by the compact and efficient wiring of biological neural circuits, specifically those observed in simple organisms like the nematode Caenorhabditis elegans. NCPs are designed to address specific challenges in robotic control systems, particularly in environments requiring real-time processing, robustness to sensory noise, and interpretability of the decision-making process. NCPs use significantly fewer neurons and connections compared to conventional deep neural networks, reducing computational overhead and enhancing real-time performance. The connections within an NCP are predominantly feedforward and sparse, mimicking the efficient wiring observed in C. elegans. This sparsity is not only biologically inspired but also contributes to the robustness and generalizability of the network. The structured and sparse nature of NCPs allows for easier analysis and understanding of how network decisions are made, an essential feature for applications in safety-critical systems.

The typical architecture of an NCP consists of four types of layers, each with a specific role:

Sensory neurons: These neurons receive input from the environment, analogous to sensory organs in biological organisms.

Interneurons: They process signals from sensory neurons and can pass information to other interneurons or command neurons.

Command neurons: Act as decision-making units that receive processed information and determine the actions to be executed.

Motor neurons: Execute the decisions by sending outputs to actuators or other systems.

NCPs are typically trained using backpropagation through time (BPTT), taking into account the dynamics of the neural model. The training involved using Adam, adaptive moment

estimation, is a variant of the gradient descent algorithm optimising the synaptic weights to minimise binary cross entropy.

The human brain is organised hierarchically, with sensory inputs processed at various levels of complexity before reaching decision-making areas like the prefrontal cortex. NCPs similarly employ a hierarchical structure where sensory neurons receive inputs, interneurons process these inputs, command neurons make decisions based on the processed information, and motor neurons execute actions. This mirrors the sensory input through processing to output execution pathways seen in the brain.

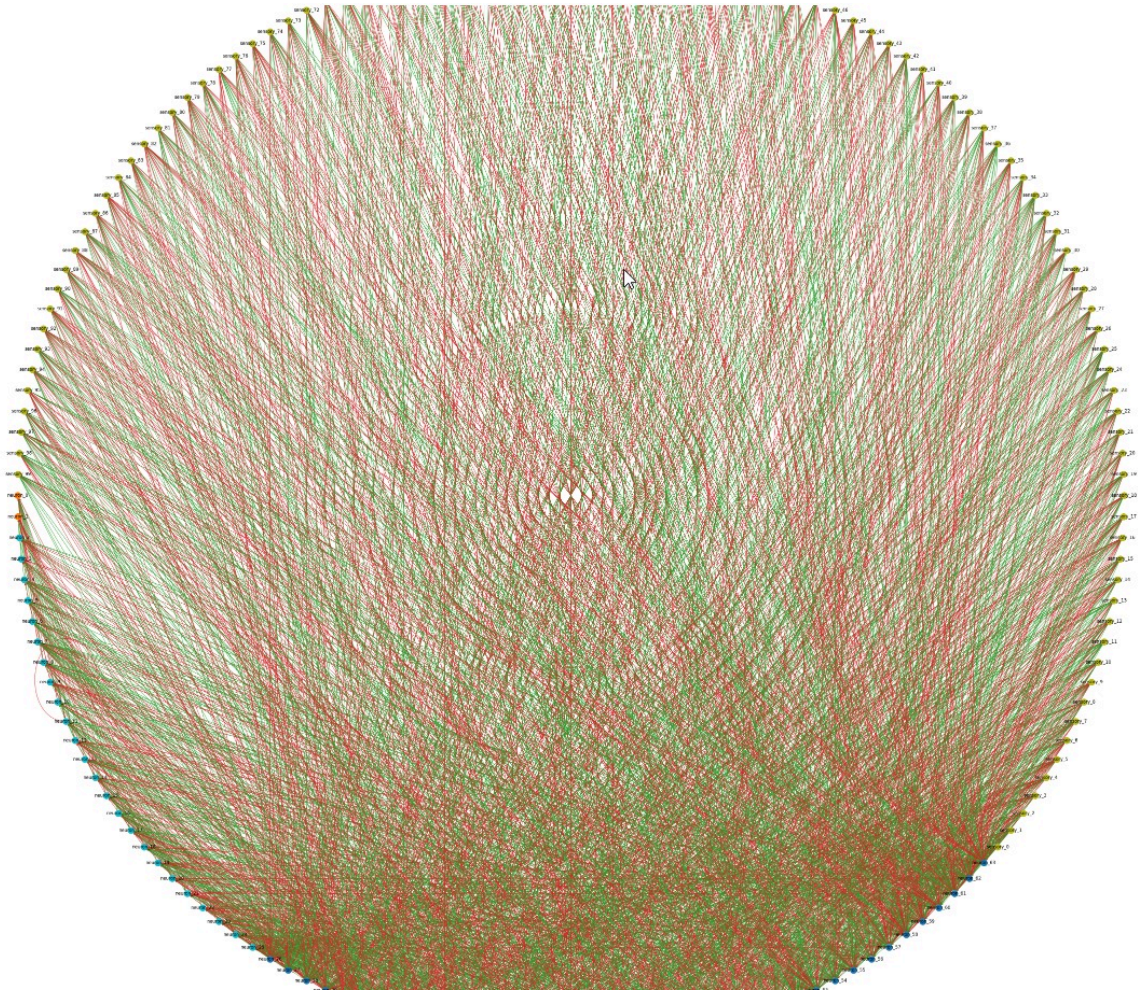

Figure 40: 1-Layer Long-Term Coding (LTC) Model Architecture

The above figure shows the 1 layer LTC model with 64 interneurons and an input of 100 features used in the project.

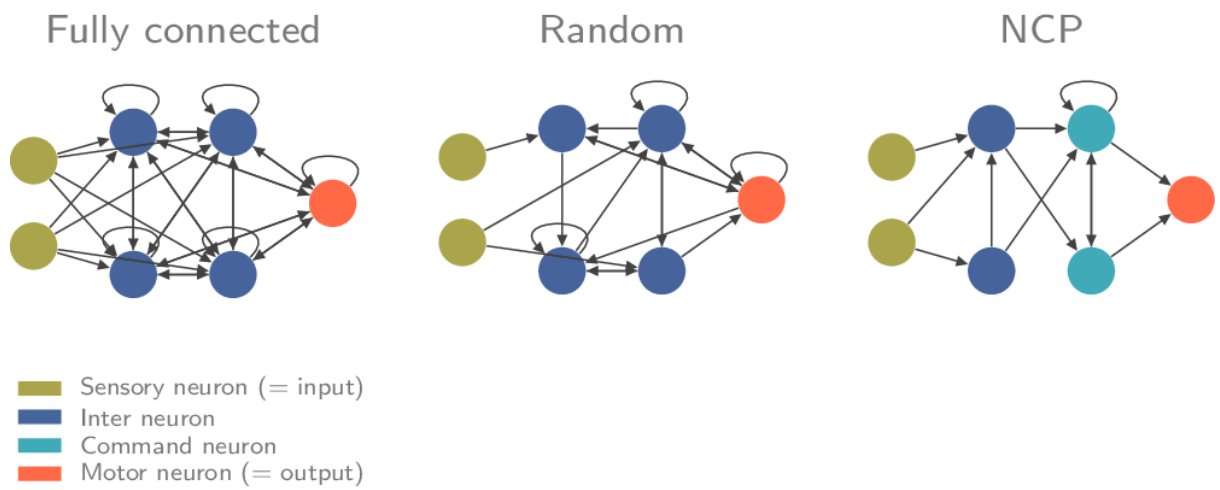

Figure 41: Simplified Neural Coding Paradigm (NCP) Diagram

The above diagram shows a simplified NCP.

4.4 Classification Results

4.4.1 Subjectivity

CountVectoriser Unigram

Model	Accuracy	F1	Precision	Recall
Naive Bayes	0.55	0.53	0.55	0.56
KNN	0.42	0.41	0.55	0.54

Table 7: CountVectoriser Unigram Performance

TF-IDF Unigram

Model	Accuracy	F1	Precision	Recall
SVM	0.56	0.54	0.58	0.59

Table 8: TF-IDF Unigram Performance

TF-IDF Unigram + BERT word embeddings

Model	Accuracy	F1	Precision	Recall
Logistic Regression	0.58	0.57	0.60	0.62

Table 9: TF-IDF Unigram + BERT word embeddings Performance

GloVe embeddings

Model	Accuracy	F1	Precision	Recall
Neural Circuit Policies With Liquid Time-Constant Network	0.71	0.83	0.71	1
CNN baseline	0.71	0.71	0.71	0.71
SCNN with population coding	0.71	0.83	0.71	1
Baseline FFN	0.71	0.71	0.71	0.71
SFFN	0.63	0.77	0.7	0.84
ESN	0.66	0.66	0.66	0.66
ESN with dot product attention	0.70	0.70	0.70	0.70
Bidirectional SLSTM	Timeout	Timeout	Timeout	Timeout

Table 10: GloVe embeddings Performance

We realise that our models were underperforming for subjectivity detection. Upon inspecting the training dataset collated from online sources, our group found that the “Neutral” labels were not accurate. There were numerous headlines that our annotators agreed that they would label as “Positive” or “Negative” but were instead labelled as “Neutral”. This is probably the reason why the models were unable to make good predictions on the self-labelled evaluation dataset. Examples of poorly labelled “Neutral” headlines are shown below:

Headlines	Online Labelled Sentiment	Our Annotator’s Sentiment
Tesla Loses Its Third General Counsel in the Course of a Year	Neutral	Negative
Congratulations to @ServiceNow on being added to the S&P 500 https://t.co/MzojJ6sPLS	Neutral	Positive

The 3 Biggest Risks to Momo’s Stock	Neutral	Negative
These Stocks Could Bounce High in January and Beyond	Neutral	Positive
After the sale, Outokumpu's share of the technology unit will be reduced to some 12-20 percent.	Neutral	Negative
Garmin fell 4.5 percent to $ 34.53 at 1:33 p.m. in New York , while Google slid 0.7 percent at $ 576.50.	Neutral	Negative
However , the production is almost entirely very labour intensive and based on small investments only .	Neutral	Negative
In this market, gaining 12.8% for the 12 months makes you the top stock-fund manager https://t.co/aVQ9SiucK7	Neutral	Positive

Table 11: Online Labelled Sentiment vs Our Annotator’s Sentiment prediction

4.4.1.1 A deeper look into cognitive models

4.4.1.1.1 Neural Circuit Policies With Liquid Time-Constant Network

Figure 42: Neural Circuit Policies With Liquid Time-Constant Network Performance

Loss: The training loss decreases over time, which is expected as the model learns from the training data. However, the test loss increases, indicating the model is not generalising well to unseen data. This divergence between training and test loss is a classic sign of overfitting.

Accuracy: The training accuracy improves significantly over time, yet the test accuracy shows very little improvement after the initial epochs. This is another sign that the model is learning to memorise the training data, rather than learning generalizable patterns.

Precision and Recall: Both precision and recall for the training set show high variance but overall increasing trends. In contrast, the test precision and recall are more stable but at lower levels. Notably, the test recall has a sharp drop early on and doesn't recover much, suggesting the model is particularly poor at identifying relevant instances in the test set.

Time: This model is relatively slow, taking about 14.4 seconds per epoch to train.

This model consists of 1 pretrained embedding layer that uses the GloVe embedding matrix, followed by the LTC model with 64 interneurons and 2 motor neurons. The output is flattened and fed to the final linear output layer with a sigmoid activation function. With this relatively simple model, the model is already experiencing severe overfitting.

4.4.1.1.2 1D CNN Benchmark

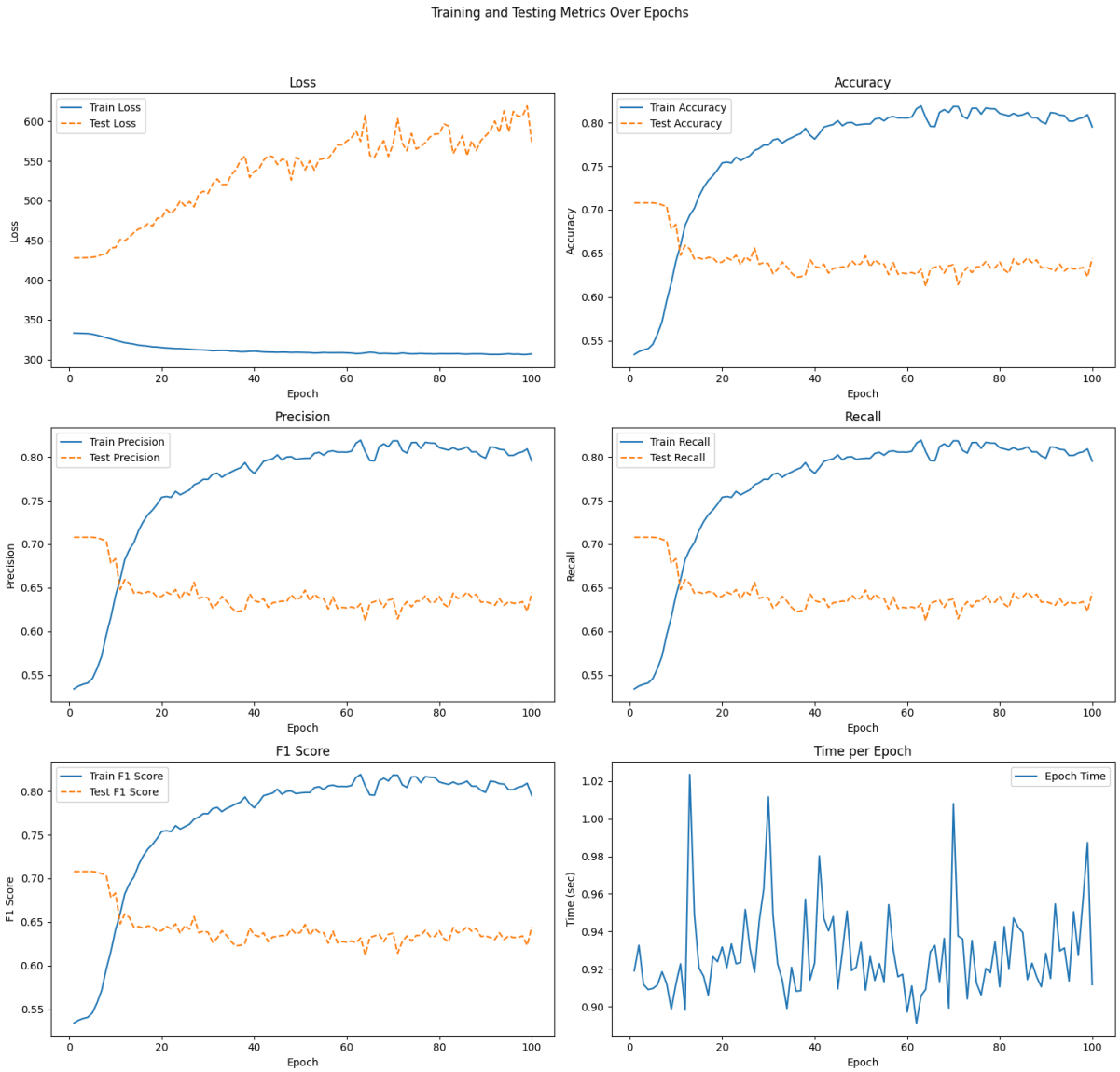

Figure 43: 1D CNN Benchmark Performance

A simple 2 layer 1D CNN was used as a benchmark, the architecture is as follows.

```
CNN(
  (embedding): Embedding(17870, 100)
  (conv1): Conv1d(100, 64, kernel_size=(3,), stride=(1,))
  (relu1): ReLu()
  (pool1): max_pool1d()
  (conv2): Conv1d(64, 32, kernel_size=(5,), stride=(1,))
```

```
  (relu2): ReLu()
  (pool2): max_pool1d()
  (fc1): Linear(in_features=96, out_features=2, bias=True)
)
```

Loss: The training loss decreases smoothly, which is good. The testing loss increases as epochs increases, indicating sever overfitting

Accuracy: The training accuracy improves consistently and stabilises around 70%, but the testing accuracy fluctuates around 40%. This significant gap indicates overfitting.

Precision, Recall, F1 Score : The metrics show an inverse relationship between the improvement over epochs in train and test data before plateauing, showing significant overfitting.

Time per Epoch: The time per epoch is relatively stable and low, which indicates good efficiency.

4.4.1.1.3 1D SCNN with LiF

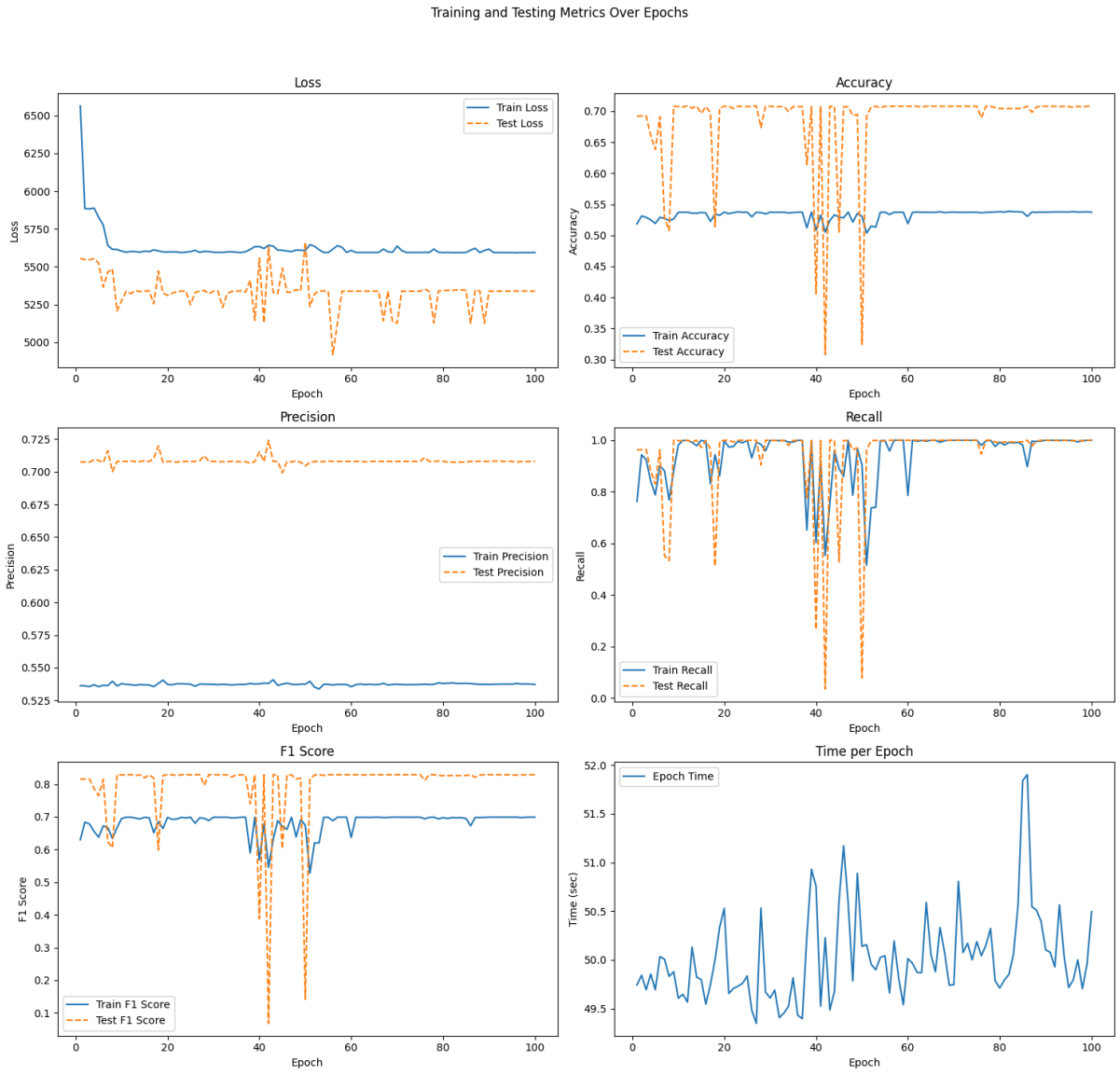

Figure 44: 1D SCNN with LiF Performance

A spiking CNN (SCNN) with population coding of 15 neurons per class and 100 timesteps for the simulation. was created with similar layers and hyperparameters to the baseline CNN. The architecture is as follows.

```
SNNCNN_pop(
  (embedding): Embedding(17870, 100)
  (conv1): Conv1d(100, 64, kernel_size=(3,), stride=(1,))
  (lif1): Leaky()
```

```
  (pool1): max_pool1d()
  (conv2): Conv1d(64, 32, kernel_size=(5,), stride=(1,))
  (lif2): Leaky()
  (pool2): max_pool1d()
  (fc1): Linear(in_features=96, out_features=30, bias=True)
  (lif3): Leaky()
)
```

Loss: The training and testing loss both decrease and then remain relatively flat with less fluctuation than the ReLU model. This could indicate a better generalisation.

Accuracy: Training accuracy increases and plateaus at about 53%, while testing accuracy increases and plateaus at about 70% but has sudden drops at least 30%. The sharp drops may be due to the optimiser being too aggressive. The testing accuracy being significantly higher than the training accuracy and the fact that the model's metrics plateaus quickly may indicate underfitting and might need more complexity.

Precision: The training precision and testing precision stays relatively stable throughout the epochs indicating that the model is underfitting

Recall: Testing recall is almost perfect, but has extreme dips which correspond to the dips in accuracy and F1 score. This shows that the model struggles with the opinionated class when there is an aggressive update. Both the training and testing recall increases at the start to a perfect score.

F1 Score: Similarly, the F1 score increases at the start to and plateaus with testing F1 score plateauing higher. This indicates that the model is underfitting and might need more complexity.

Time per Epoch: The training time is less stable and higher than the ReLU model, which indicates less computational efficiency.

The SCNN mode does not seem to overfit as the number of epochs increases and the metrics stabilises despite the number of epoch increasing, indicating that the model lacks complexity.

4.4.1.1.4 Bi-directional LSTM with LiF activation

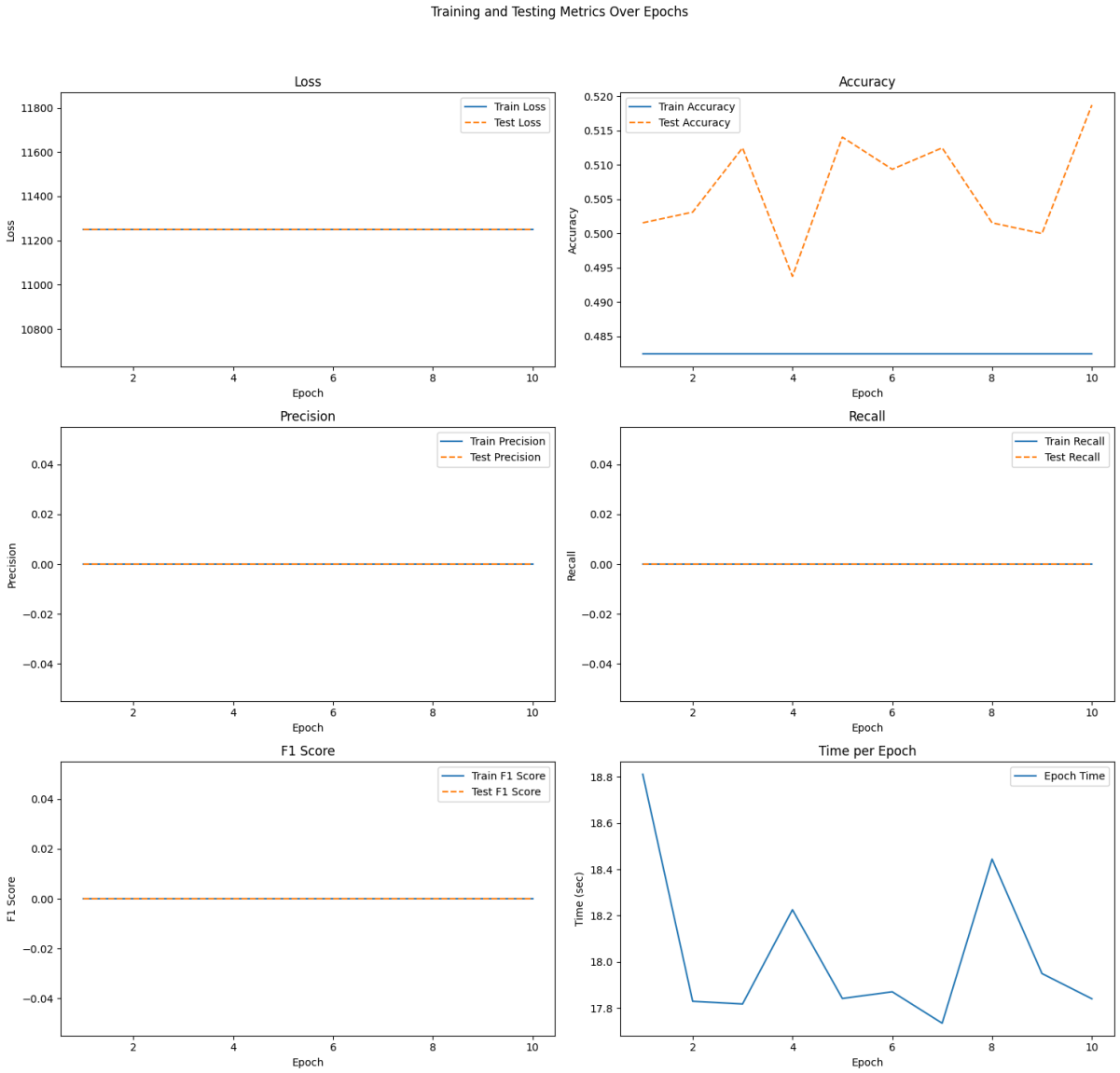

Figure 45: Bi-directional LSTM with LiF activation Performance

With an embedding layer, a Bi-directional LSTM with LiF neurons and a final output layer with LiF neurons, and population coding, of 15 neurons per class. This spiking LSTM (SLSTM), was trained on a much smaller dataset of 500 hand labelled samples. The SLSTM returned predicted the NEGATIVE class constantly and took about 10 times longer per epoch compared to the SCNN with 500 training samples. This is due to the fact that LSTM

processes the data sequentially, and is unable to parallelize. Thus we did not proceed to the full test data, due to limitations in computational resources.

4.4.1.1.5 Baseline FFN

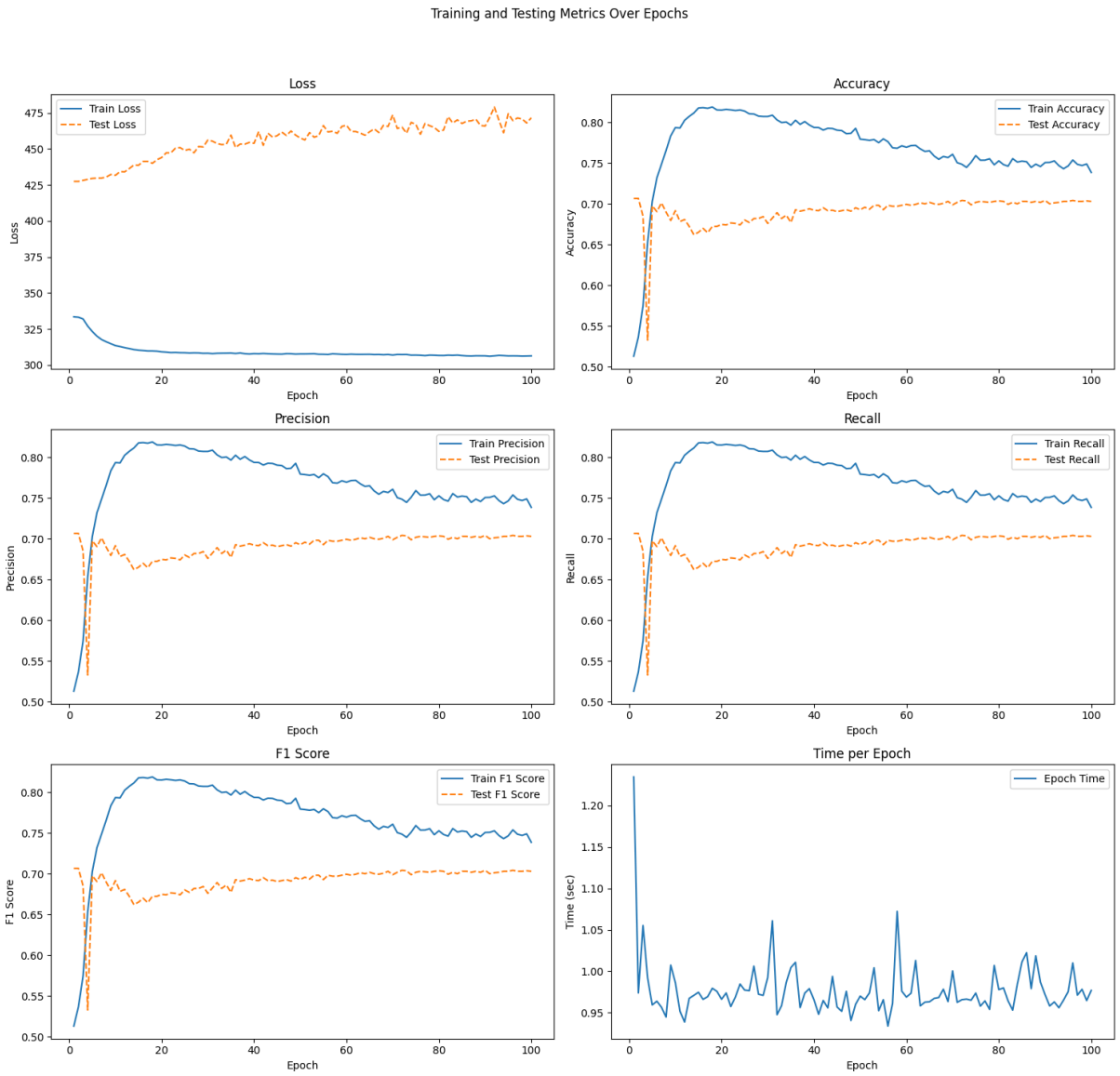

Figure 46: Baseline FFN Performance

The model consists of 2 hidden layers of size 64 and 32, with dropout layers with 0.8 dropout rate after max pooling the output of the hidden layers . The hidden layers use the ReLu activation function. The test loss diverges from the training loss indicating overfitting. Despite overfitting, the model did not learn anything from the data.

4.4.2.1.6 SFFN

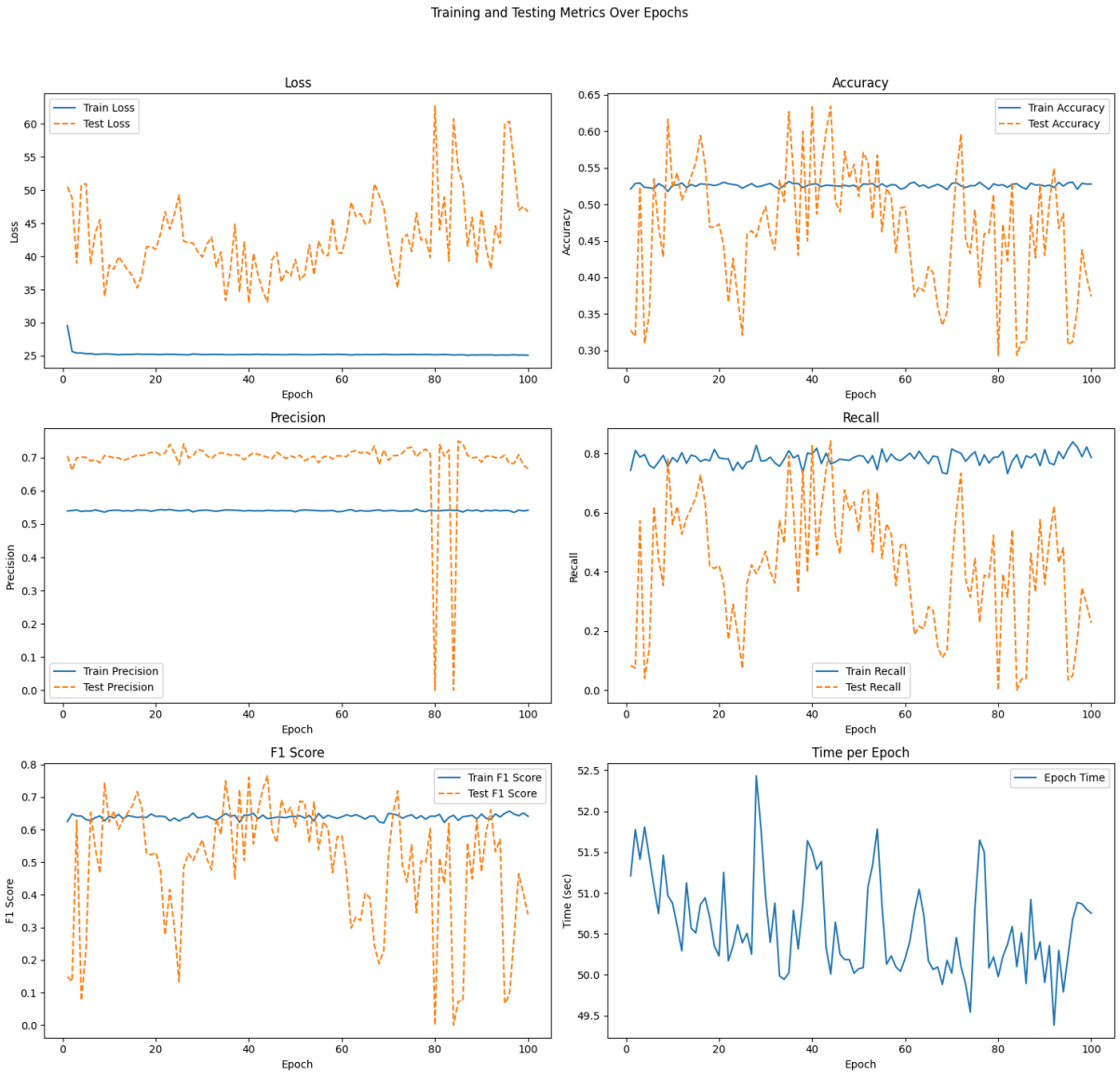

Figure 47: SFFN Performance

The SFFN's architecture is similar to the Baseline FFN with the activation function being changed to LiF activation function. Despite almost no change in the training metrics, the testing metrics swing wildly, indicating that this model is mostly memorising the training data rather than learning to generalise from it. When exposed to new, unseen data in the testing set, it fails to perform consistently, leading to the observed fluctuations in the testing metrics. This might be due to the FFN architecture being highly unsuited for text classification with GloVe embeddings.

4.4.1.1.7 ESN

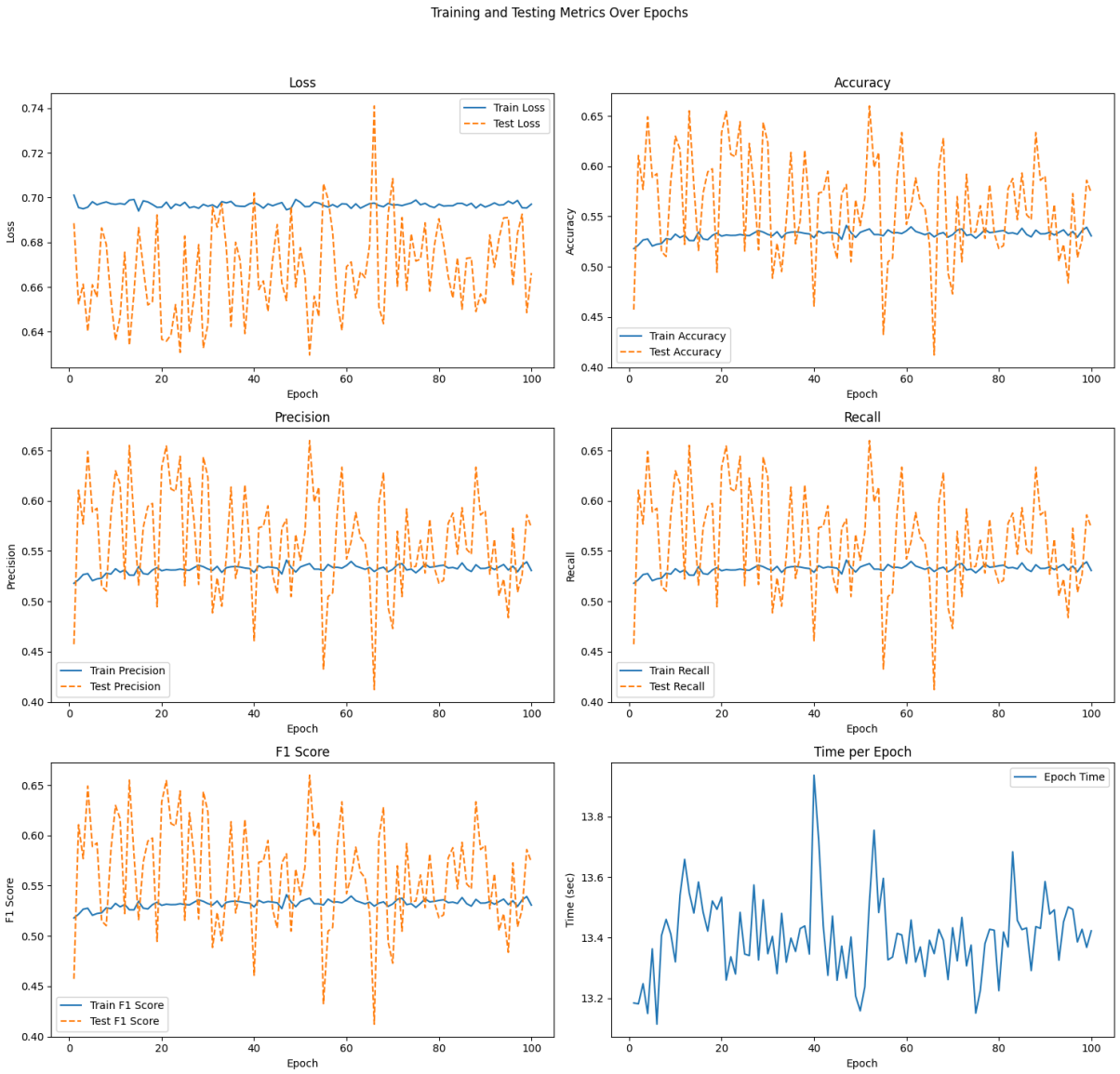

Figure 48: ESN Performance

The ESN with attention consists of a Reservoir with 2 layers. The model did not seem to learn anything and the test metrics swung wildly, despite the training metrics' much lower variance. The model does not seem to have learnt anything as shown by the almost constant train metrics, this might be due to the fact that the reservoir does not update the individual neurons' weights.

4.4.1.1.8 ESN with attention

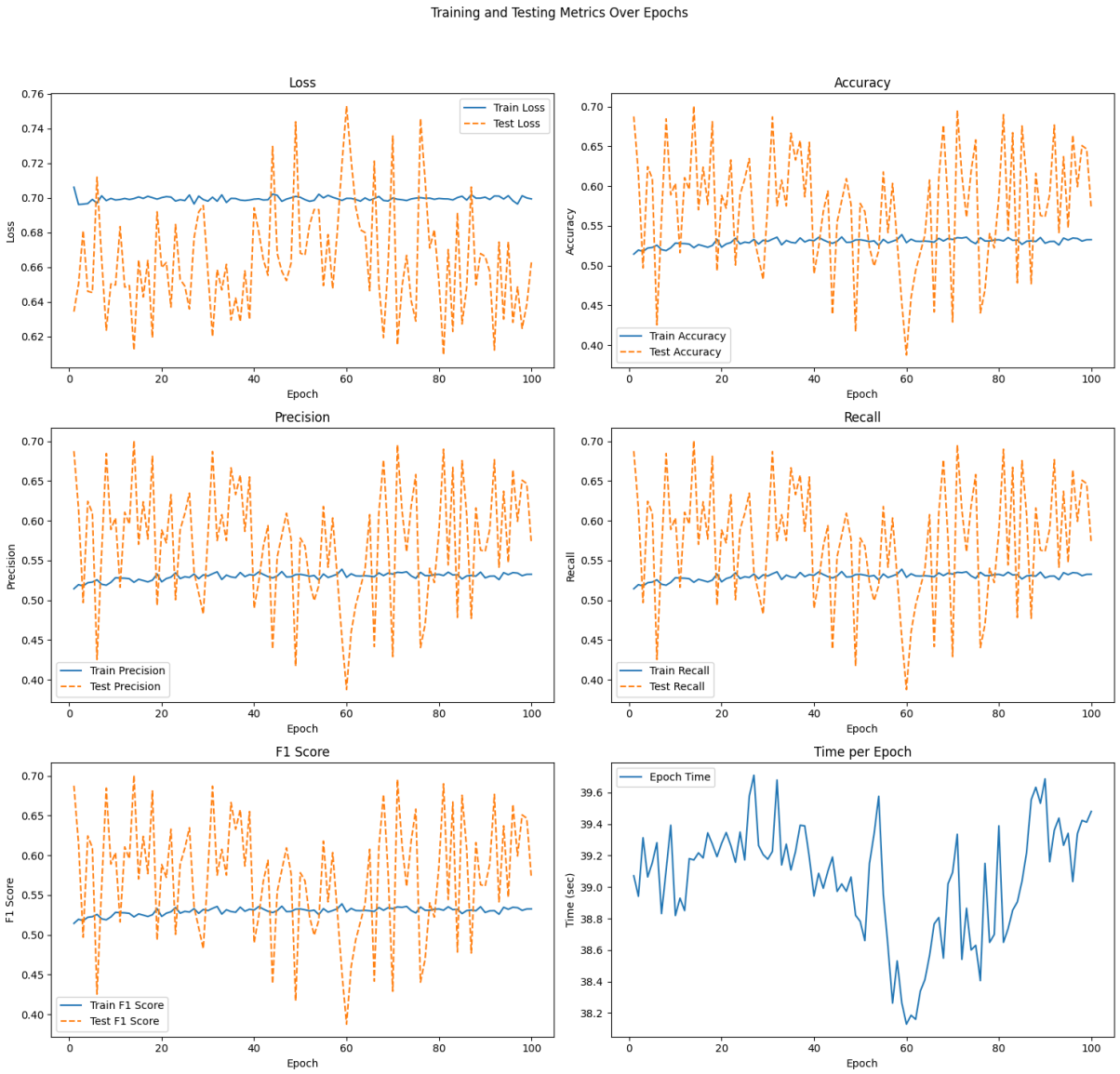

Figure 49: ESN with attention Performance

The ESN with attention consists of a Reservoir with 2 layers, followed by a dot product attention layer. The output of both layers are concatenated together and normalised. This combined output is passed to the Reservoir with 2 layers, followed by an attention layer, and the output of these 2 layers are then concatenated together before the final output layer. The addition of attention with the add and norm layers seems to perform slightly better than the base ESN but it still suffers the same problem that there is almost no learning.

4.4.2 Polarity

CountVectoriser Unigrams

Model	Accuracy	F1	Precision	Recall
Naive Bayes	0.70	0.70	0.71	0.70
KNN	0.50	0.46	0.50	0.50

Table 12: CountVectoriser Unigrams Performance

TF-IDF Unigrams

Model	Accuracy	F1	Precision	Recall
SVM	0.73	0.72	0.73	0.72
Ensemble - Hard Voting (Random Forest, SVM, Logistic Regression)	0.74	0.73	0.74	0.76
Ensemble - Soft Voting (Random Forest, SVM, Logistic Regression)	0.75	0.74	0.76	0.75
Ensemble - Stacking (Random Forest, SVM)	0.76	0.76	0.76	0.76

Table 13: TF-IDF Unigrams Performance

TF-IDF Unigrams and Bigrams

Model	Accuracy	F1	Precision	Recall
Ensemble - Hard Voting (Random Forest, SVM, Logistic Regression)	0.74	0.73	0.76	0.74
Ensemble - Soft Voting (Random Forest, SVM, Logistic Regression)	0.74	0.74	0.75	0.74
Ensemble - Stacking (Random Forest,	0.75	0.75	0.75	0.75

SVM)				

Table 14: TF-IDF Unigrams and Bigrams Performance

TF-IDF Unigrams + BERT word embeddings

Model	**Accuracy**	**F1**	**Precision**	**Recall**
Logistic Regression	0.79	0.78	0.79	0.79
Ensemble - Soft Voting (Random Forest)	0.58	0.50	0.72	0.58
Ensemble - Stacking (Random Forest, Logistic Regression)	0.78	0.78	0.78	0.78

Table 15: TF-IDF Unigrams + BERT word embeddings Performance

TF-IDF Unigrams and Bigrams + BERT word embeddings

Model	**Accuracy**	**F1**	**Precision**	**Recall**
Logistic Regression	0.77	0.77	0.79	0.77

Table 16: TF-IDF Unigrams and Bigrams + BERT word embeddings Performance

TF-IDF Unigrams, Bigrams and Trigrams + BERT word embeddings

Model	**Accuracy**	**F1**	**Precision**	**Recall**
Logistic Regression	0.78	0.78	0.78	0.78

Table 17: TF-IDF Unigrams, Bigrams and Trigrams + BERT word embeddings Performance

Our models for detecting polarity presented a better performance.

The results also showed that including the use of BERT word embeddings help to improve the accuracy, particularly our logistic regression model.

GloVe embeddings

Model	Accuracy	F1	Precision	Recall
Neural Circuit Policies With Liquid Time-Constant Network	0.5	0.67	0.5	1
CNN baseline	0.53	0.53	0.53	0.53
SCNN with population coding	0.5	0.67	0.5	1
Baseline FFN	0.52	0.52	0.52	0.52
SFFN	0.51	0.67	0.5	1
ESN	0.52	0.52	0.52	0.52
ESN with dot product attention	0.53	0.53	0.53	0.53
Bidirectional SLSTM	Timeout	Timeout	Timeout	Timeout

Table 18: GloVe embeddings Performance

However the neural networks inspired by cognitive functions, with GloVe embeddings, performed much worse at polarity classification.

4.4.2.1 A deeper look into cognitive models

4.4.2.1.1 Neural Circuit Policies With Liquid Time-Constant Network

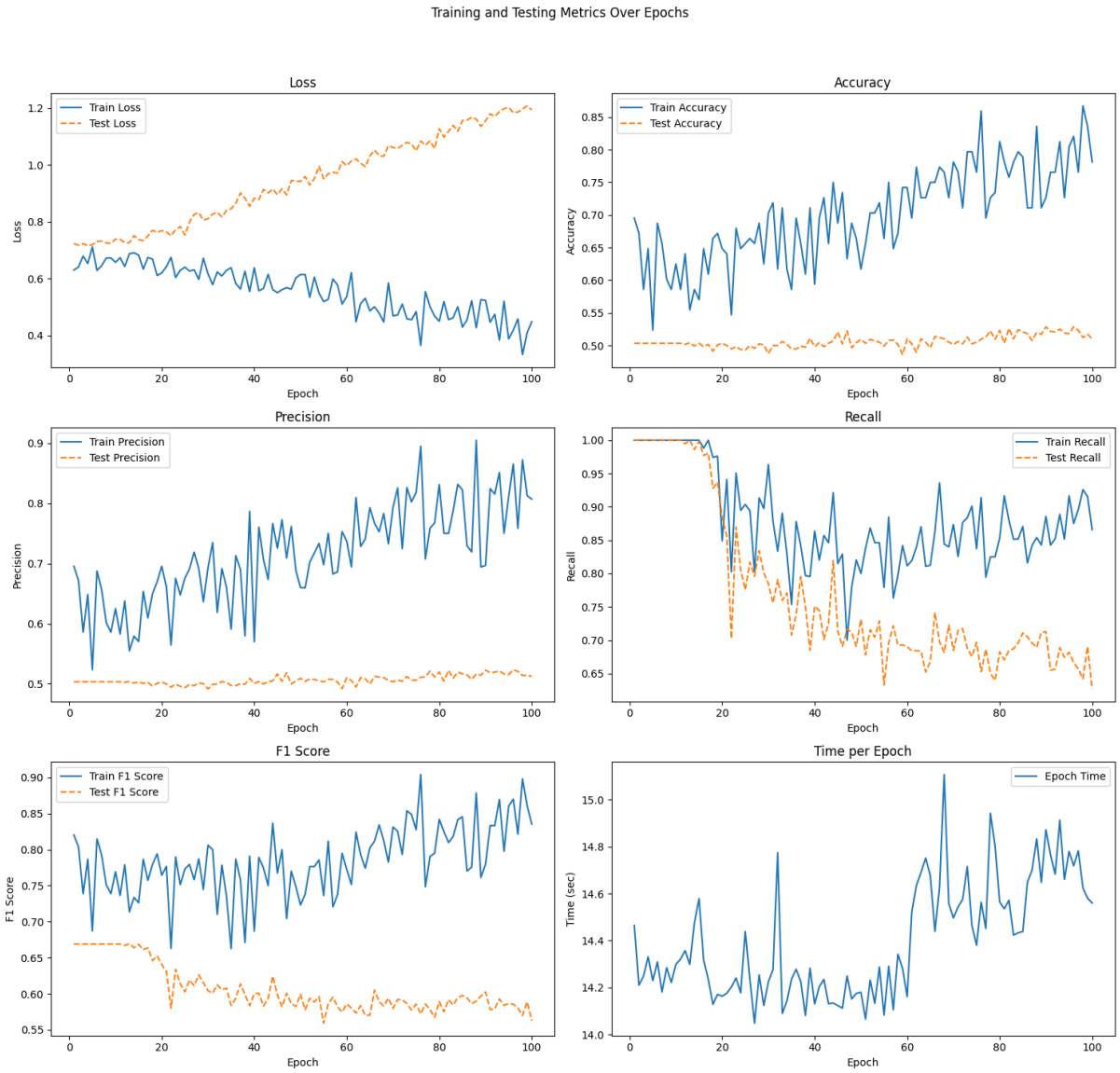

Figure 50: Neural Circuit Policies With Liquid Time-Constant Network Performance

The architecture is similar to the one used for subjectivity.Similarly the model only learns noise and overfits.

4.4.2.1.2 1D CNN Benchmark

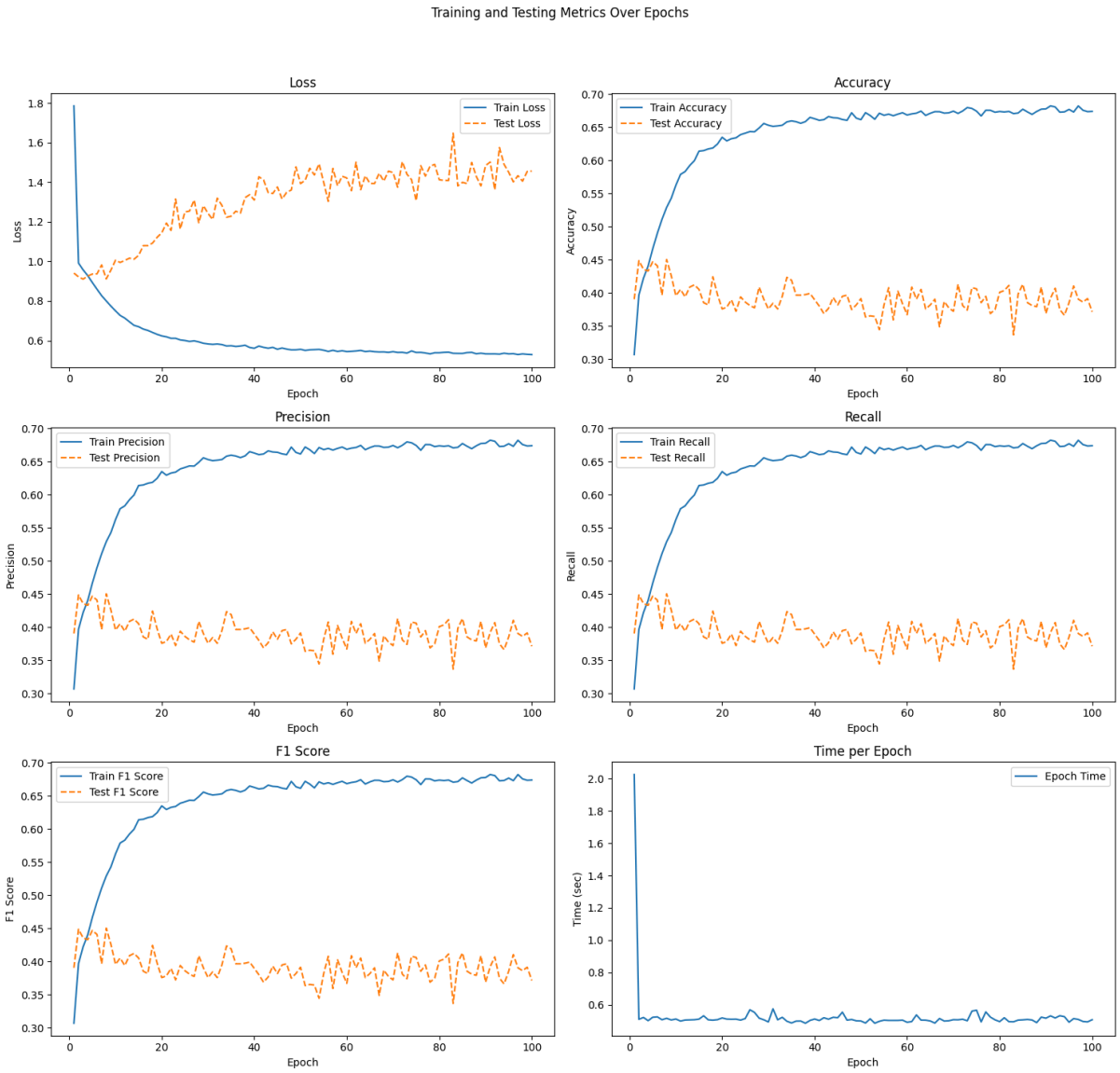

Figure 51: 1D CNN Benchmark Performance

This model's architecture is the same as the 1D CNN used in subjectivity classification.

Loss: The training loss consistently decreases, indicating learning. However, the testing loss is somewhat volatile but generally maintains a consistent average, which could indicate significant overfitting as the model performs better on the training data compared to the testing data.

Accuracy, Precision, Recall and F1 Score: There is a significant gap between training and testing metrics, with training metrics improving substantially while testing metrics does not.

Time per Epoch: The time per epoch is stable and low, which is good for training efficiency.

4.4.2.1.3 1D SCNN with LiF

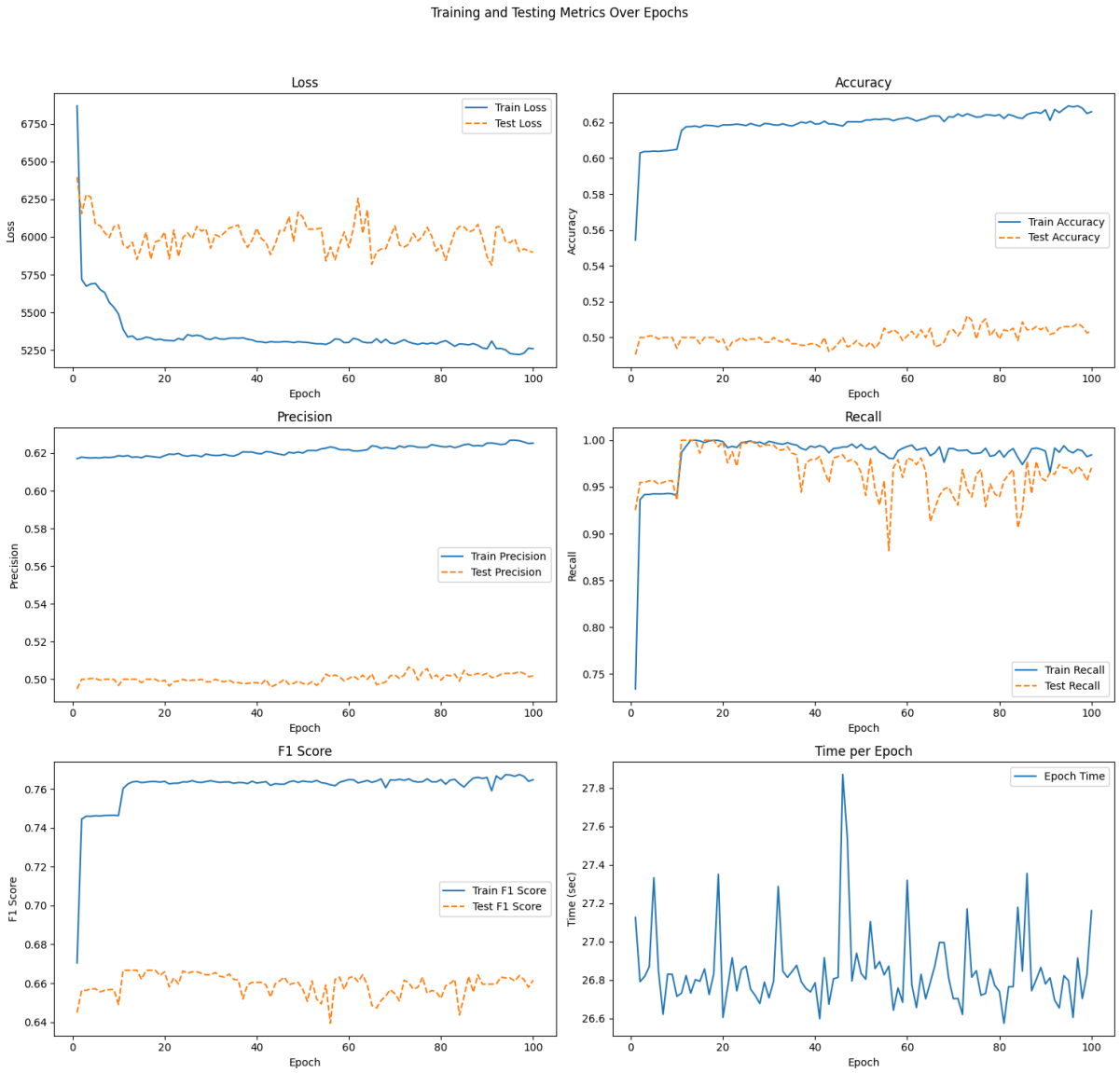

Figure 52: 1D SCNN with LiF Performance

This model's architecture is the same as the 1D SCNN used in subjectivity classification.

Loss: The training loss decreases sharply and then levels off, while the testing loss decreases slightly and then fluctuates. This could be an indicator of overfitting, similar to the 1D CNN with ReLU activation.

Accuracy: The training accuracy improves significantly over epochs and stabilises, whereas the testing accuracy shows slight improvement.

Precision: Training precision shows slight improvement, but test precision remains flat and low, suggesting the model has difficulty generalising the positive predictions.

Recall: The recall is almost perfect for the training set, which is unusual and could indicate overfitting. The testing recall's fluctuation increases as the number of epochs increases, indicating overfitting.

F1 Score: Training F1 score shows improvement, while testing F1 score is lower and volatile.The testing F1 score's fluctuation increases as the number of epochs increases, indicating overfitting.

Time per Epoch: There is significant volatility in the time per epoch, which may indicate inconsistent computational load or inefficiency in training.

The SCNN performs better than the benchmark CNN on the test set, however, this comes at the cost of performance. The time per epoch for the spiking CNN with LiF neurons is about 55.4 times higher and more inconsistent, suggesting that it might be less efficient in terms of computation time compared to the 1D CNN with ReLU activation.

4.4.2.1.4 Baseline FFN

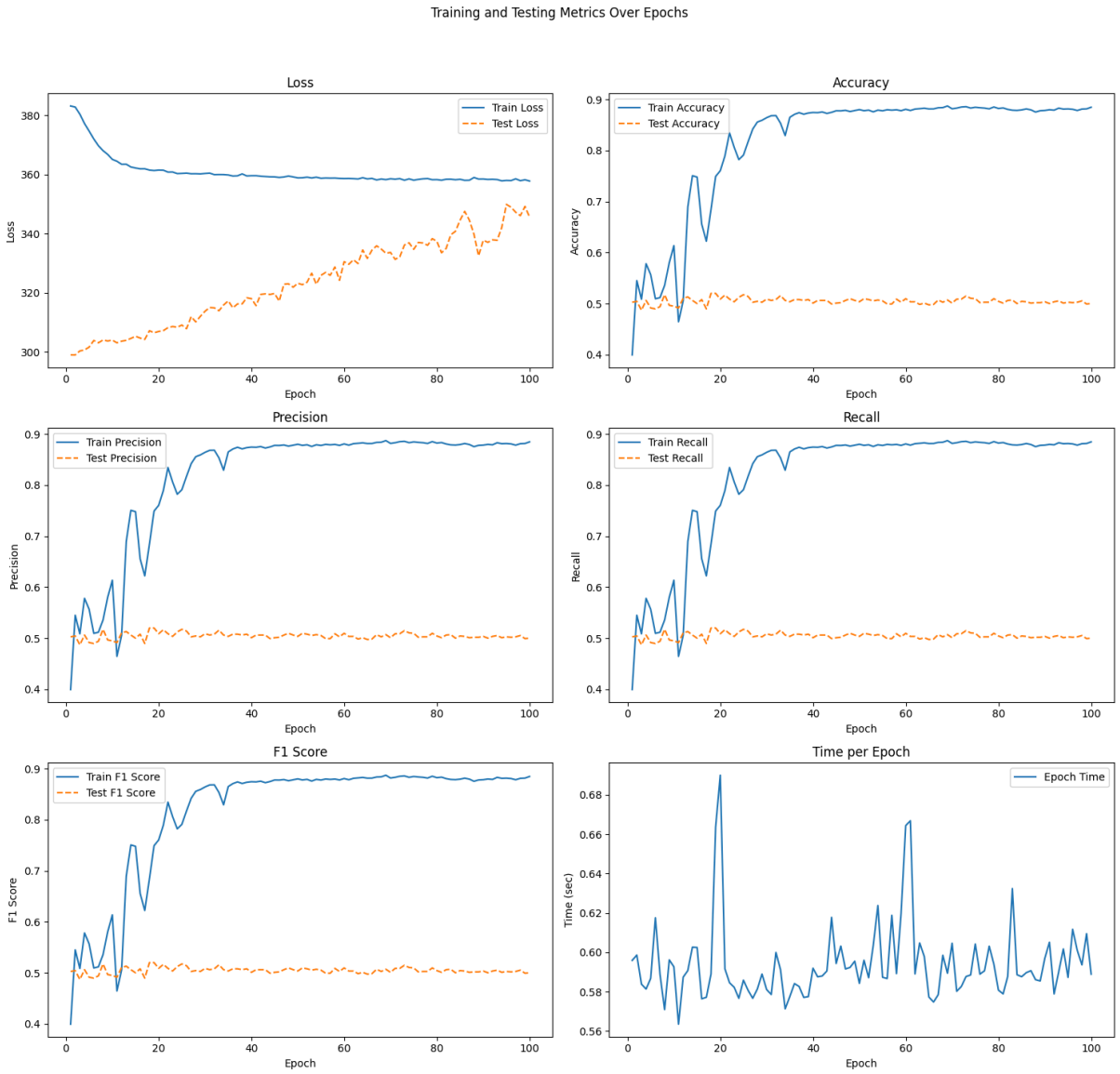

Figure 53: Baseline FFN Performance

The model's architecture is the same as the Baseline FFN used for subjectivity classification. Despite overfitting, the model did not learn anything from the data as the test Accuracy, Precision, Recall and F1 Score remains constant while the loss increases .

4.4.2.1.5 SFFN

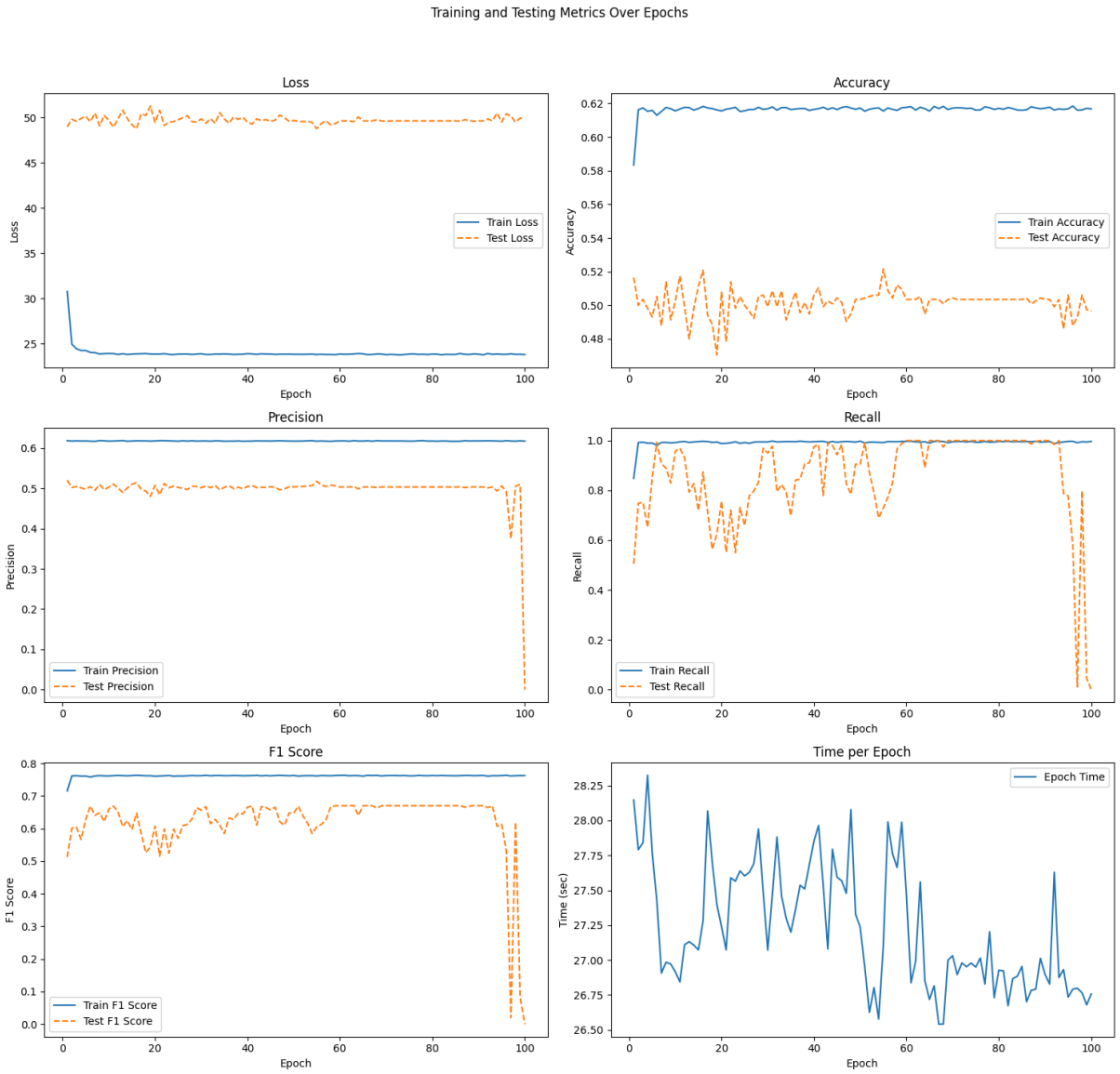

Figure 54: SFFN Performance

The model consists of 2 hidden layers of size 64 and 32, with dropout layers with 0.8 dropout rate, similar to the baseline. Despite overfitting, the model did not learn anything from the data. Compared to the baseline FNN, the SFFN's recall and F1 score is better.

4.4.2.1.6 ESN

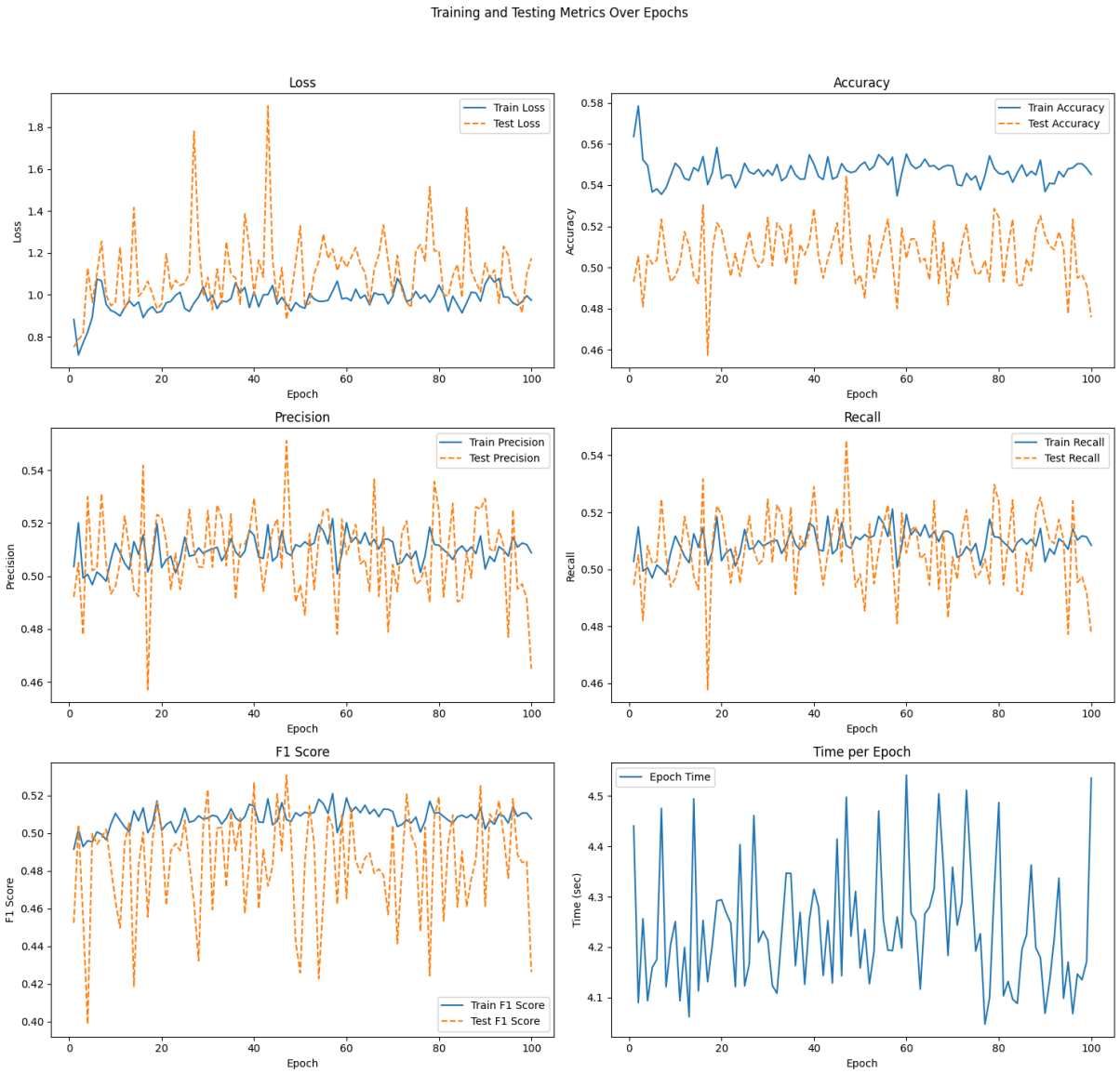

Figure 55: ESN Performance

The ESN's architecture is the same as for the subjectivity task. The model did not seem to learn anything and the test metrics swung wildly.

4.4.2.1.7 ESN with attention

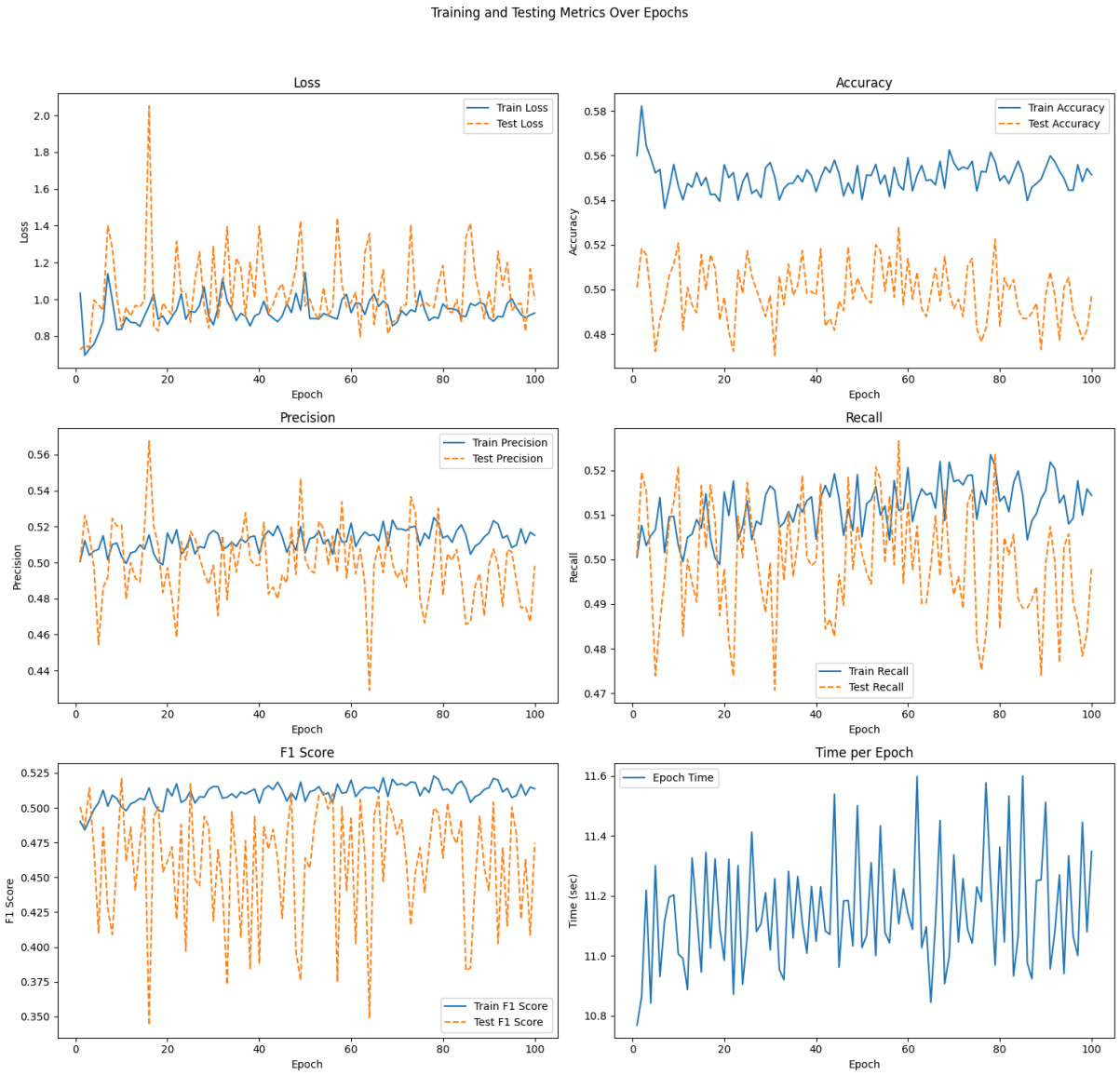

Figure 56: ESN with attention Performance

The ESN with attention's architecture is the same as for the subjectivity task. The model did not seem to learn anything and the test metrics swings wildly, despite the significantly lower variance of the training metrics. As with the Subjectivity task, adding the attention layers with the add and norm layers shows a slight improvement over the base ESN.

4.5 Enhancements

4.5.1 Ensemble Classification

Ensemble classification is a type of machine learning technique used to increase model accuracy by integrating different models in order to create a more powerful and robust model. There are different types of ensemble methods. In this project, we used the voting and stacking ensemble method.

Voting methods are useful when each individual model has their own strengths and weaknesses. In the voting method, the multiple classification model makes a prediction on the sentiment, and these predictions are considered as a “vote”. The final prediction from the model will be the majority “vote” from the different models. There are also 2 types of voting. In hard voting, the final prediction is determined by the majority vote, in other words, the class with the most votes is chosen as the final prediction. In soft voting, instead of counting the majority vote, each model is assigned a confidence score to each class label. The final prediction is then determined by calculating the average confidence score across all the models and the class label with the highest score is selected.

The other ensemble method is the stacking method. Several base models are first trained on the same dataset and are used to make predictions on the validation set. The predictions made on the validation set are stacked together to create a new feature matrix, with each row representing a data point and each column representing the predictions made by one of the base models. Lastly, a meta-model is trained on the feature matrix created and the ground truth labels and combines the predictions of the base model to make a final prediction.

The initial SOTA SVM model had an accuracy of 72.4%. After using the voting method by adding Random Forest and Logistic Regression, the accuracy increased slightly to 74.6% and 73.7% for soft and hard voting respectively. Soft voting performed better likely due to its ability to provide better insights by utilising the confidence scores rather than simply just having a majority vote. On the other hand, the stacking method using Random Forest and SVM increased the accuracy to 75.7%.

In comparison, our TF-IDF Unigram + BERT word embedding model originally had an accuracy of 78.5% while using Logistic Regression. After using the soft voting method of Random Forest instead, the accuracy decreased to 58.1%. Also, when the ensemble stacking method was used on both Random Forest and Logistic Regression, the accuracy decreased to 77.7%

4.6 Discussion

4.6.1 Random accuracy test

```
import random

# Generate a random number from the list of rows to choose the row in which the model is tested on
random_int = random.randint(0, (len(df_9k)-1))
print("Random row chosen:", random_int)
```

```
Random row chosen: 890
```

```
import pandas as pd

#Set the maximum column width to display the full content
pd.set_option('display.max_colwidth',None)

random_row = pd.DataFrame(df_9k_sentiLabelled.iloc[888]).transpose() #shows the 890th row
print("Random row:")
print(random_row)

```

```
Random row:
                                                        Headline  \
888  Microsoft Co. (NASDAQ:MSFT) Shares Sold by Marshall Financial Group LLC

            Source     Posted  \
888  ETF DAILY NEWS  26/3/2024

                                                                                                                                                                  Description  \
888  Marshall Financial Group LLC lowered its holdings in Microsoft Co. (NASDAQ:MSFT - Free Report) by 3.1% in the fourth quarter, according to its most recent Form 13F filing ...

                                                                                                       Link  \
888  https://www.etfdailynews.com/2024/03/26/microsoft-co-nasdaqmsft-shares-sold-by-marshall-financial-group-llc/

    FINAL Sentiment
888        NEGATIVE
```

```
import random
# Sentiment generated by random chance

# Define the three sentiments
Senti = ['POSITIVE','NEGATIVE','NEUTRAL']

# Select random text string
random_senti = random.choice(Senti)

#Print the randomly selected Sentiment
print("Randomly selected Sentiment:",random_senti)
```

```
Randomly selected Sentiment: POSITIVE
```

Figure 57: Code snippet of Random accuracy test

Performing a random accuracy test on our stock sentiment classification task enables us to establish a baseline performance level for our sentiment analysis model. In this test, we randomly selected a row, in this case, row 980. We have previously labelled the remaining data with our sentiment analysis model so we would just be taking the result from that row. It was found that the sentiment is 'NEGATIVE' when "Marshall Financial Group LLC lowered its holdings in Microsoft Co. …" ,showing that this sentiment prediction by the model is accurate. We then randomly chose a sentiment to assign to this text without considering the headline content to show the difference between random chance and our model. The randomly chosen sentiment was 'POSITIVE' which was the opposite of what the headline implied.

This test showed that our sentiment classifier performs way better than random chance. If the classifier's accuracy is not better than random chance, it indicates that the model is not effectively capturing the sentiment in the text and is essentially performing no better than random guessing. This would suggest that the model needs improvement or that the task is particularly challenging.

4.6.2 Performance metrics

4.6.2.1 Speed of prediction

```
## Make predictions using the trained model
##-for logistic regression only
start_time1 = time.time()
y_pred_9k = loaded_neuopi_model.predict(df_combined_9k_filled_reordered)
#y_pred_9k = logreg_model1.predict(df_combined_9k_filled_reordered)
end_time1 = time.time()
time_all1 = end_time1 - start_time1

#print(y_pred_9k)
#print("Time taken for all predictions: ", time_all1, " seconds")
print("Time taken for one prediction: ", time_all1/num_rows, " seconds")
```

```
Time taken for one prediction:  0.0002964261534471161  seconds
```

Figure 58: Code snippet of Speed of prediction using neutral/opinionated model

```
## Make predictions using the trained model
##-for logistic regression only
start_time2 = time.time()
y_pred_9k1 = loaded_posneg_model.predict(df_magic)
#y_pred_9k = logreg_model2.predict(df_filtered)
end_time2 = time.time()
time_all2 = end_time2 - start_time2

#print(y_pred_9k1)
#print("Time taken for all predictions: ", time_all2, " seconds")
print("Time taken for one prediction: ", time_all2/num_rows, " seconds")
```

```
Time taken for one prediction:  7.607619123064217e-05  seconds
```

Figure 59: Code snippet of Speed of prediction using positive/negative model

```
print("Time taken for one prediction: ", (time_all1 + time_all2)/num_rows,
      "seconds")
```

```
Time taken for one prediction:  0.00037250234467775826 seconds
```

Figure 60: Code snippet of Total Speed of prediction

Time Taken/s	Predictions
0.0037250234467775826	1
1	268

Table 19: Prediction Speed

Our logistic regression model based on TF-IDF Unigrams and BERT word embeddings can make 268 predictions per second. Given the simplicity of logistic regression, the classification task can be performed very quickly with little computational resources used.

4.6.2.2 Scalability

Logistic regression entails calculating a linear combination of input features, then applying a sigmoid function to the outcome. This linear computation can be efficiently performed using matrix operations, which can be parallelized and optimised for high performance with extensive datasets. Consequently, our logistic regression model shows promise for scalability with large datasets. Given that news is to be scraped on the minute/hour basis, SGD Classifier can be adopted as well to deal with the massive growing amount of data which may be too

much for the Logistic Regression model's optimization algorithm, L-BFGS, to handle. L-BFGS is more computationally expensive and requires more memory, because it needs to approximate the Hessian matrix and stores more search information. Furthermore, sklearn's L-BFGS only supports L2 Regularization while SGD is more flexible.

5. Conclusion

In this project, an information retrieval system for financial news is built. The system is an invaluable tool for investors, financial analysts, and researchers alike, offering them access to timely news and sentiment analysis based on their input queries. The system also enhances user experience through various innovations like timeline search, multilingual search, enhanced search and auto query suggestor for spelling mistakes.

State of the art sentiment classification models like SVM, logistic regression with BERT word embeddings are also trained on large amounts of online data for subjectivity and polarity detection on the system's news articles. Moreover, by further integrating sentiment classification techniques, such as ensemble and brain inspired classification models, the system is able to provide users with valuable insights into the sentiment underlying news headlines, aiding them in making informed decisions in the stock market.

6. Submission Link

YouTube Link:
https://youtu.be/OA1dNFSjDNE

Data File Link:
https://drive.google.com/file/d/1XNbC52KCoHM1J2ekV68N2XLR5mF66KDd/view?usp=drive_link

Code File Link:
https://drive.google.com/file/d/14sEPFmz9hqrTLJu1l0TOW-pjsGQw_4in/view?usp=drive_link

Backup Link:
https://drive.google.com/drive/folders/1XrqRD0Xnf_rlDQ5jAmT6ZLCLgdjd7lJ2