MODIFIED WORD REPRESENTATION VECTOR BASED SCALAR WEIGHT FOR CONTEXTUAL TEXT CLASSIFICATION

ABBAS SALIIMI LOKMAN

DOCTOR OF PHILOSOPHY

UNIVERSITI MALAYSIA PAHANG
AL-SULTAN ABDULLAH

UNIVERSITI MALAYSIA PAHANG AL-SULTAN ABDULLAH

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name	: ABBAS SALIIMI BIN LOKMAN
Date of Birth	: 13 JANUARY 1984
Title	: MODIFIED WORD REPRESENTATION VECTOR BASED SCALAR WEIGHT FOR CONTEXTUAL TEXT CLASSIFICATION
Academic Session	: SEMESTER II 2023/2024

I declare that this thesis is classified as:

☐ CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997)*
☐ RESTRICTED	(Contains restricted information as specified by the organization where research was done)*
☑ OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang Al-Sultan Abdullah reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang Al-Sultan Abdullah
2. The Library of Universiti Malaysia Pahang Al-Sultan Abdullah has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

[illegible] [illegible]

(Student's Signature)	(Supervisor's Signature)
	Mohamed Ariff bin Ameedeen
New IC/Passport Number	Name of Supervisor
Date: 24 June 2024	Date: 24 June 2024

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

MAKLUMAT PANEL PEMERIKSA PEPERIKSAAN LISAN

(for Faculty of Computing student only)

Tesis ini telah diperiksa dan diakui oleh
This thesis has been checked and verified by

Nama dan Alamat Pemeriksa Dalam *Name and Address Internal Examiner*	:	Profesor Madya Dr. Mohd Nizam Bin Mohmad Kahar Faculty of Computing Universiti Malaysia Pahang Al-Sultan Abdullah
Nama dan Alamat Pemeriksa Luar *Name and Address External Examiner*	:	Prof. Ts. Dr. Ali Bin Selamat Malaysia – Japan International Institution of Technology Universiti Teknologi Malaysia
Nama dan Alamat Pemeriksa Luar *Name and Address External Examiner*	:	Prof. Dr. Salwani Binti Abdullah Faculty of Information Science and Technology Universiti Kebangsaan Malaysia

Disahkan oleh Penolong Pendaftar IPS
Verified by Assistant Registrar IPS

Tandatangan: *Signature*	Tarikh: *Date*

Nama:
Name

اونيۏرسيتي مليسيا ڤهڠ السلطان عبدالله
UNIVERSITI MALAYSIA PAHANG
AL-SULTAN ABDULLAH

SUPERVISOR'S DECLARATION

We hereby declare that We have checked this thesis and in our opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Doctor of Philosophy in Computer Science.

[illegible]

____

(Supervisor's Signature)

Full Name	: Dr. Mohamed Ariff bin Ameedeen
Position	: Associate Professor
Date	: 24 June 2024

[illegible]

____

(Co-supervisor's Signature)

Full Name	: Ts. Sr Dr. Ngahzaifa binti Ab. Ghani
Position	: Senior Lecturer
Date	: 24 June 2024

اونيۏرسيتي مليسيا ڤهڠ السلطان عبدالله
UNIVERSITI MALAYSIA PAHANG
AL-SULTAN ABDULLAH

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang Al-Sultan Abdullah or any other institutions.

[illegible]

(Student's Signature)

Full Name	: Abbas Saliimi bin Lokman
ID Number	: PCC17013
Date	: 24 June 2024

MODIFIED WORD REPRESENTATION VECTOR BASED SCALAR WEIGHT FOR CONTEXTUAL TEXT CLASSIFICATION

ABBAS SALIIMI LOKMAN

Thesis submitted in fulfillment of the requirements

for the award of the degree of

Doctor of Philosophy

Faculty of Computing

UNIVERSITI MALAYSIA PAHANG AL-SULTAN ABDULLAH

JUNE 2024

ACKNOWLEDGEMENTS

Alhamdulillah, I am grateful to God Almighty, for without His blessings and guidance, this study would not have been completed.

Next, I would like to express my heartfelt gratitude to my supervisors, Associate Professor Dr. Mohamed Ariff Ameedeen and Sr Dr. Ngahzaifa Ab. Ghani, for their invaluable insights, guidance, comments, and patience throughout my journey in completing this study. I would also like to extend my thanks to all the funders: Skim Latihan Akademik Bumiputra (SLAB) and Fundamental Research Grant Scheme (FRGS No. FRGS/1/2018/ICT02/UMP/02/12) from Ministry of Higher Education, as well as the Fundamental Research Grant (RDU No. RDU220315) from Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA), for their unwavering financial support for this study and research.

To my beloved wife, Nurliyana Mohd Johari, I am deeply grateful for your constant presence through both hardships and moments of joy. This study would not have been completed without your steadfast support by my side. To my cherished children, Abdul Muhaimin ‘Adnin and Awfa Tasnim, you are the light of my soul. Thank you for being the source of joy in my everyday life. To my dear mother, Mariani Mat, your unwavering strength has been a constant anchor in my universe. I am truly thankful for your enduring support. Also, to all my family members, thank you for your dua and well wishes.

Last but not least, I would like to express my heartfelt appreciation to all my friends and colleagues who have stood by me, no matter how small the matter. Your presence has enriched my life in ways beyond measure, and I cherish every second of it.

Thank you all.

ABSTRAK

Tesis ini mengkaji klasifikasi teks kontekstual, yang merupakan proses mengkategorikan data teks ke dalam kelas atau kategori yang berbeza berdasarkan maknanya dalam konteks yang diberikan. Komponen utama dalam proses ini adalah representasi perkataan melalui vektor untuk interpretasi pengkomputeran. Praktis semasa ialah menggunakan Model Bahasa Besar atau *Large Language Model* (LLM) untuk menghasilkan vektor representasi perkataan yang terkontekstual. Penghasilan ini diperolehi melalui pra-latihan LLM pada korpora yang luas bagi membolehkannya memahami corak bahasa dan konteks yang rumit. Untuk klasifikasi teks kontekstual, LLM pra-latihan tersebut akan melalui satu lagi proses latihan menggunakan data yang diberi label khusus untuk klasifikasi. Proses latihan kedua ini dipanggil penalaan halus. Walaupun pendekatan pra-latihan dan penalaan halus pada ketika ini dianggap yang paling optimal dalam bidang ini, ia berhadapan cabaran yang besar dari segi kos pengkomputeran yang diperlukan. Ini adalah kerana jumlah parameter yang perlu dilatih di dalam LLM adalah sangat besar, menyebabkan kos latihan menjadi sangat tinggi. Tambahan lagi, walaupun LLM pra-latihan boleh menghasilkan vektor representasi perkataan yang terkontekstual, ia tidak mempunyai fleksibiliti untuk mengubah nilai semantik vektor-vektor tersebut di luar LLM. Untuk menutup jurang ini, kajian ini menyusun metodologi penyelidikan lima fasa bagi mencadang dan menilai algoritma yang membolehkan modifikasi luaran vektor perkataan yang dihasilkan oleh LLM menggunakan nilai skalar sebagai faktor tumpuan. Untuk menilai algoritma ini, vektor perkataan yang telah diubahsuai dibandingkan dengan vektor perkataan asal yang dihasilkan oleh LLM bagi menganalisis kesan modifikasi berkenaan dalam konteks yang difokuskan. Selain itu, eksperimen klasifikasi teks kontekstual juga dijalankan bagi menilai prestasi vektor perkataan yang telah diubahsuai tersebut dalam proses klasifikasi yang diperlukan. Untuk eksperimen ini, vektor perkataan yang telah diubahsuai akan digunakan sebagai input untuk melatih model Pembelajaran Mesin atau *Machine Learning* (ML) bagi tugas klasifikasi teks dengan matlamat untuk membangunkan model ML yang mempunyai bilangan parameter yang jauh lebih kecil dari LLM. Eksperimen ini bertujuan untuk menilai keberkesanan vektor perkataan yang telah diubahsuai dalam tugas klasifikasi teks kontekstual dengan menggunakan pendekatan pengkomputeran yang lebih efisien. Berdasarkan hasil eksperimen yang diperolehi, ia menunjukkan bahawa algoritma yang dibangunkan dapat mengubah vektor perkataan asal yang dihasilkan oleh LLM bagi mencerminkan konteks yang dikehendaki dalam proses klasifikasi teks kontekstual yang berkenaan. Ini dapat dilihat melalui hasil eksperimen yang memperolehi skor lebih tinggi daripada skor rujukan. Metrik penilaian yang digunakan dalam eksperimen berkenaan ialah Ketepatan, Kejituan, Kecetusan, dan Skor F1, dengan Ketepatan dan Skor F1 berfungsi sebagai metrik utama. Hasil penilaian metrik berkenaan menunjukkan peningkatan yang ketara, dengan model ML ujian mencapai skor ketepatan terbaik sebanyak 0.571, peningkatan sebanyak 46% dari skor rujukan, dan skor F1 terbaik sebanyak 0.727, peningkatan sebanyak 30% dari skor rujukan. Secara keseluruhan, tesis ini membentangkan lima sumbangan utama dalam bidang kajian iaitu algoritma bagi mengubahsuai vektor perkataan, set data klasifikasi kontekstual baharu bernama QCoC, pengelas soalan yang efisien berdasarkan algoritma *feed-forward neural network*, potensi untuk memindahkan kerja yang dibentangkan kepada domain lain, dan implikasi praktikal kerja yang dibentangkan kepada kes-kes di mana sumber pengkomputeran adalah terhad atau mahal.

ABSTRACT

This thesis investigates contextual text classification, which is the process of categorising textual data into different classes or categories based on its meaning within a given context. Central to this process is the representation of words through vectors for computational interpretation. Current practices employ Large Language Models (LLMs) to generate contextualised word representation vectors, achieved through pre-training the LLM on vast corpora that enables it to grasp intricate language patterns and context. For contextual text classification, the pre-trained LLM is further train on classification-specific labeled data in a process called fine-tuning. Although this approach is currently considered the most optimal in the field, it poses a notable challenge due to the substantial demand for computing resources stemming from the vast number of trainable parameters in LLMs. Furthermore, although pre-trained LLMs can generate contextualised word representation vectors, they lack the flexibility to modify the semantic significance of these vectors outside of the LLM, necessitating fine-tuning for the modification of word vectors. To bridge this gap, a five-phase research methodology is structured to propose and evaluate an algorithm enabling the external modification of LLM-generated word vectors using scalar values as the focus weightage. To validate this algorithm, the modified word vectors are compared with original LLM-generated word vectors to evaluate their reflection of the intended context. In addition, a contextual text classification experiment is conducted using benchmarked datasets to assess the performance of the modified word vectors in the targeted classification task. For this experiment, the modified word vectors serve as input to train a Machine Learning (ML) model for the text classification process, aiming for the developed ML model to have a significantly smaller parameter count. This experiment aims to determine the effectiveness of the modified word vectors in contextual text classification tasks, utilizing a more computationally efficient approach. Based on the acquired results, the experiments reveal that the modified word vectors algorithm can effectively alter original LLM-generated word vectors to reflect intended contexts and can outperform baseline scores in contextual text classification tasks. Evaluation metrics including Accuracy, Precision, Recall, and F1 score are employed in the evaluation process, with Accuracy and F1 score serving as primary metrics. The evaluation showcases significant improvements, with the test ML model achieving a best accuracy score of 0.571, a 46% increase from the baseline, and a best F1 score of 0.727, a 30% increment from the baseline. Overall, this thesis presents five contributions: the proposed modified word vectors algorithm, the new contextual classification dataset named QCoC, the efficient question-type classifier based on the feed-forward neural network algorithm, the potential transferability of the presented work to other domains, and the practical implications of the presented work towards cases where computational resources are limited or costly.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

b	neural network's bias
e^{z_i}	exponential function for an input vector
e^{z_j}	exponential function for an output vector
f	neural network's activation function
FN	False Negative (wrong prediction of the truth class)
FP	False Positive (wrong prediction of other than the truth class)
gw	general g-weight value for each token (in scalar format)
i	index for any given array
K	number of classes in the given classification dataset
TN	True Negative (correct prediction of the negative/false class)
TP	True Positive (correct prediction of the positive/truth class)
v	vector embedding for each token/word
vs	vector embedding for the input sentence
w_i	neural network's weight
wv	Weighted fixed-length vector
x_i	neural network's input
$\vec{z}$	input vector from previous neural network's nodes

LIST OF ABBREVIATIONS

CoQA	A Conversational Question Answering Challenge
DA	Discriminant Analysis
DNLM	Deep Neural Language Model
FF-NN	Feed-Forward Neural Network
GRU	Gated Recurrent Unit
IR	Information Retrieval
kNN	k-Nearest Neighbours
LLM	Large Language Model
LM	Language Model
LSTM	Long-Short Term Memory
ML	Machine Learning
MRC	Machine Reading Comprehension
NB	Naïve Bayes
NLI	Natural Language Inference
NLP	Natural Language Processing
NMT	Neural Machine Translation
NN	Neural Network
QA	Question-Answering
QCoC	Question Classification of CoQA
QuAC	Question Answering in Context
QTC	Question Type Classification
RNN	Recurrent Neural Network
RQ	Research Question
SQuAD	The Stanford Question Answering Dataset
SVM	Support Vector Machines
TF	Term Frequency
TREC	The Text Retrieval Conference
TT	Total Token
UT	Unique Token
USE	Universal Sentence Encoder
WR	Word Representation

CHAPTER 1

INTRODUCTION

1.1 Background

Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on the interaction between humans and computers using natural human language (Khurana et al., 2023). Its primary goal is to enable machines to understand, interpret, and generate human language in a way that is both meaningful and contextually relevant. NLP encompasses a wide range of tasks, including language translation, sentiment analysis, text summarisation, text classification, and information retrieval, among others (Qin et al., 2023).

One crucial aspect of NLP is the representation of words in a form that computers can understand and process (Naseem et al., 2021). Traditional methods often represented words as discrete symbols, but more recent approaches employ word vectors, which capture semantic relationships and contextual information (Jiao & Zhang, 2021; Sezerer & Tekir, 2021; Uymaz & Metin, 2022; Patil et al, 2023). Word representation vectors transform words into high-dimensional numerical vectors, positioning them in a multi-dimensional space based on their meaning and context (Salim & Mustafa, 2022; Wadud et al., 2022). Weightage in word representation vectors refers to the significance assigned to each dimension within the vector space. This significance reflects the importance of certain semantic features or contextual nuances associated with a word (Mundotiya et al., 2022; Ghosal & Jain, 2022). The process of learning these vectors involves training models on large corpora of text, where words with similar meanings or contexts end up being closer together in the vector space (Liu et al., 2023).

In NLP, classification is a common problem where the goal is to categorize text data into predefined classes or categories (Gasparetto et al., 2022; Li et al., 2022a). Word representation vectors play a crucial role in solving classification problems by capturing the semantic meaning of words and their relationships. These vectors allow NLP models to understand the context and nuances within a text, enabling more accurate and context-aware predictions (Abubakar et al., 2022; Dogra et al., 2022). Classification models often utilize machine learning algorithms such as support vector machines, decision trees, or neural networks (Wahba et al., 2023). The input to these models is typically the word representation vectors derived from the text data. The model learns to associate certain patterns in the vector space with specific classes, enabling it to make predictions on new, unseen text data (Allammary, 2022; Li et al., 2022a; Pham et al., 2023). In essence, the interplay between word representation vectors and machine learning algorithms forms the backbone of effective text classification, enabling systems to navigate the intricacies of language and derive meaningful insights from diverse textual contexts (Da Costa et al., 2023).

New approaches to text classification have shifted towards the fine-tuning of Large Language Model (LLM) (Dogra et al., 2022; Qassim et al., 2022; Bilal & Almazroi, 2023; Zhang et al., 2024). This approach involves using advanced models like BERT (Devlin et al., 2018) and GPT (Brown et al., 2020), pre-trained on vast datasets to understand complex language patterns and context (pre-training is a process before fine-tuning). In fine-tuning, these LLMs are exposed to task-specific labeled data, allowing them to adapt to specific tasks while retaining their broader linguistic knowledge. This method has proven effective not only in various natural language understanding applications but also in achieving good performance in text classification tasks (Min at al., 2023). The key idea behind fine-tuning LLMs is the smart use of transfer learning principles (Azunre, 2021; Bharadiya, 2023). This approach leverages the intrinsic linguistic knowledge already embedded in the LLMs through pre-training, significantly diminishing the requirement for extensive amounts of task-specific labeled data. In essence, this methodology compliment the strengths of pre-training with the fine-tuning process, resulting in a framework that exhibits good performance in various text classification tasks such as sentiment analysis, document categorization, and others (Kici et al., 2021; Qasim et. Al, 2022; Ameer et al., 2023; Onita, 2023; Wadud et al., 2023).

1.2 Problem Statement and Research Gap

Leveraging the fine-tuning of LLMs for text classification presents a promising path to attain state-of-the-art results in this NLP downstream task (Dogra et al., 2022; Mars, 2022). However, this approach comes with a notable challenge that is the demand for substantial computing resources (Sharir et al., 2020; Church et al., 2021; Fernandez et al., 2023). Fine-tuning process entails adjusting the pre-trained LLM parameters on task-specific data, necessitating extensive computational power and storage capabilities due to the vast numbers of parameters in LLM. This poses a significant hurdle for smaller research teams, educational institutions, or others with limited access to high-performance computing resources, hindering their ability to leverage the benefits of fine-tuned LLMs for text classification.

While the demand for computational resources presents a significant challenge, an additional research gap exists in the inflexibility to externally modifying the weight of word vectors without the necessity to modify, re-train, or fine-tune the original word embeddings model such as the LLM (Dogra et al., 2022; Incitti et al., 2023; Patil et al., 2023; Worth, 2023). In many instances, word vectors from the embeddings model are directly used without any modification for the NLP downstream tasks, including text classification. Examples of such cases include Sabri et al. (2022), who directly employs word vectors generated from TF-IDF (Jalilifard et al., 2021), word count (Kowsari et al., 2019), and Word2Vec (Mikolov et al., 2013; Church, 2016) embeddings models to compare results for Arabic text classification; Zhou, who utilizes a TF-IDF embeddings model to extract words with sufficient weight, then generates word vectors using Word2Vec embeddings model, and feeds the word vectors into a Convolutional Neural Network – Long-Short Term Memory (CNN-LSTM) model for text classification (Zhou, 2022a); Soni et al., who uses the Word2Vec embeddings model to generate word vectors from multiple n-grams tokens, which are later fed into a CNN model for the text classification process (Soni et al., 2023); Umer et al., who generated the word vectors using FastText embeddings model (Joulin et al., 2016; Dharma et al., 2022) for their text classificatiom problem (Umer et al., 2023); and Hossain et al., who propose a text classification framework that takes word vectors from various embedding models, compares them, and selects the best one for input in the machine learning model for the text classification process (Hossain et al., 2023).

As mentioned, a research gap exists in the inflexibility of externally modifying the weight of word vectors without the necessity to modify, re-train, or fine-tune the original word embeddings model. Despite this limitation, several attempts to externally modify the word vector exist, typically involving weightage in vector format (usually concatenating both vectors to produce the final word vector value). In practical terms, although the original embeddings model remains unaltered, a new model is required to generate the weightage, constituting another word vector value and incurring additional computing resources, which are relatively similar to re-training the base model. Examples of this practice within text classification tasks include Badri et al., who combine FastText (Joulin et al., 2016) and GloVe (Pennington et al., 2014) embeddings models for classifying hate speech text (Badri et al., 2022); Zhou et al., who combine GloVe and Word2Vec (Mikolov et al., 2013) for text sentiment classification (Zhou et al., 2022b); and Liu et al., who concatenate BERT (Devlin et al., 2018) and LDA (Blei et al., 2003) embeddings models to produce word vectors containing both word and topic contextual information for text classification tasks (Liu et al., 2023).

Modifying the weight of word vectors using a singular scalar value presents a notable challenge in NLP field (Apidianaki, 2023). The inherent difficulty arises from the intricate and multifaceted nature of representing words with vector values. Word vectors encapsulate complex semantic relationships, capturing the contextual nuances and intricacies of language usage. Attempting to uniformly modify these vectors with a scalar value encounters challenges due to the varying importance of individual words in different contexts (Bollegala & O'Neill, 2022; Xu et al., 2023). Words may carry diverse connotations, dependencies, and degrees of importance depending on their usage within sentences or documents. Consequently, a uniform scalar modification may overlook these subtleties, potentially leading to oversimplification or distortion of the original semantic information encoded in the word vectors. Moreover, the intricate interplay between words in a given context makes it challenging to devise a universal scalar adjustment that equally and meaningfully impacts all word vectors. In essence, the challenge lies in finding a balance that allows for meaningful modification of word vector weights while respecting the nuanced and context-dependent nature of language. Researchers and practitioners in NLP are continually exploring innovative approaches to address this challenge and enhance the adaptability of word vectors to better suit specific tasks and contexts (Patil et al., 2023; Incitti et al., 2023; Johnson et al., 2024).

1.3 Aim, Hypothesis and Research Questions

This research aims to bridge the identified research gap by proposing a new algorithm capable of modifying the weight of a word representation vector through an external scalar weight value. The resulting output from this algorithm will subsequently serve as input for a Machine Learning (ML) model, facilitating the execution of text classification task. In the context of a case study, this research aims to construct a new text classification dataset that highlights distinctions in contextual representation. Overall, this research hypothesizes that *the word representation vector derived from the LLM can be altered using an external scalar weight, which can later be used as input for an ML model to perform text classification task.*

To test the formulated hypothesis and address the research aim, the following research questions and objectives are defined:

1. How can an algorithm effectively incorporate external scalar weights into word representation vectors to enhance context understanding in a contextual text classification problem?

2. What criteria should be considered in the creation of a text classification dataset to emphasise differences in context representation?

3. How does the developed algorithm, incorporating external scalar weights, perform when applied to contextual text classification task?

1.4 Research Objectives

1. To develop a new algorithm that incorporate an external scalar weight into the word representation vector.

2. To develop a new text classification dataset that emphasises differences in context representation.

3. To evaluate the developed algorithm using a Machine Learning (ML) model in the contextual text classification problem.

1.5 Research Scopes

The scope of this research is delimited by the following constraints:

1. The proposed algorithm will only be tested in the English language. Other natural human languages might produce different results.

2. The proposed algorithm will only be tested using specific hardware and software setup. Different setups might produce different results.

3. Based on the literature, specific LLM, case study and ML model will be selected for this study. Implementing the proposed method with different LLMs, case studies and ML models might produce different results.

1.6 Thesis Organisation

1. Chapter 1 presents the background of this research, including a brief discussion on word representation and text classification in Natural Language Processing (NLP), the problem statement and research gap, and finally the aim, hypothesis, research questions, research objectives and research scopes, forming the foundation for this study.

2. Chapter 2 provides a comprehensive review of the existing literature surrounding contextual text classification, focusing on key areas defined in the problem statement and research gap for this study. A gap analysis is also provided at the beginning of this chapter to guide the flow of the literature topics. Additionally, the direction of this research is discussed at the end of this chapter to guide the structuring of the research methodology in the next chapter.

3. Chapter 3 outlines and discusses the methodology of this research, consisting of five interconnected phases. Within these phases, several design approaches are explained and several algorithms are proposed, accompanied by discussions on the expected results from each phase leading towards the next. Overall, this chapter outlines the five-phase research methodology through a flowchart and general descriptions, then delves into detailed discussions on the activities undertaken for each phase.

4. Chapter 4 discuss in details the results gathered from all experiments and testings that have been explained in the previous chapter. Before discussing the results, each phases in the research methodology is being recap with their respective expected results are explained. Then each phase's result is being discussed in details with discussion of overall findings towards this research objectives as the final analysis towards the overall results of the undergo research methodology.

5. Chapter 4 discusses in detail the results gathered from all experiments and tests explained in the previous chapter. Before delving into the results, each phase of the research methodology is briefly reviewed, along with explanations of their respective expected outcomes. Subsequently, the results of each phase are thoroughly examined, with an overarching discussion of the findings pertaining to the research objectives and research questions, serving as the final analysis of the overall results of the research methodology undertaken.

6. Chapter 5 concludes this thesis with a discussion on constraints and limitations, contributions, threats to validity, and future works for this research. In the section on constraints and limitations, the bounds within which this research is conducted are acknowledged, recognizing its limited scope. The section on contributions outlines five major contributions from this research, including a new algorithm, a new dataset, and other related implementations in other domains. The threats to validity section then discusses potential challenges to the validity of this research. Lastly, the section on future works outlines five potential avenues for further expansion of this research beyond its defined scopes

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter aims to provide a comprehensive review of the existing literature related to contextual text classification, with a particular focus on addressing key gaps and challenges mentioned in previous chapter. In essence, leveraging fine-tuned Large Language Models (LLMs) has emerged as a promising approach to achieve state-of-the-art results in contextual text classification tasks, yet significant hurdles exist, particularly concerning accessibility and flexibility. The demand for substantial computational resources poses a barrier to the widespread adoption of fine-tuned LLMs, limiting their utilization by smaller research teams and educational institutions. Furthermore, a critical gap persists in the inflexibility to externally modify the weight of word vectors without necessitating modifications to the original embeddings model such as the LLM, hindering the customization and adaptability of models for specific tasks and contexts. Additionally, challenges in uniformly modifying word vector weights with scalar values highlight the intricate and context-dependent nature of language representation. Overall, this literature review seeks to explore these gaps in the existing research landscape and lay the groundwork for addressing them through the proposed research framework. Ultimately, the goal is to devise methods and algorithms that enhance the accessibility, flexibility, and effectiveness of LLM-based text classification systems.

2.2 Gap Analysis

Building upon the problem statement and research gap discussed in the previous Chapter 1, this section will delineate the identified gaps into three main areas, each corresponding to the defined research questions and objectives 1, 2, and 3, respectively.

GAP 1: Incorporating external scalar weights into word representation vectors. Addressing the research question of how algorithms can effectively incorporate external scalar weights into word representation vectors to enhance context understanding in contextual text classification requires overcoming existing limitations in modifying word vector weights. Current literature highlights a gap in methods that enable seamless integration of external scalar weights into word vectors without necessitating modification, re-training, or fine-tuning of the original embeddings model. Existing approaches often involve computationally intensive processes or fail to adequately capture the nuanced contextual information required for accurate text classification. To explore this area further, this chapter will discuss literature surrounding word representations in vector space, particularly focusing on the weight calculation method and LLM, which is the current predominant word representation generation method.

GAP 2: **Defining criteria for creating text classification datasets emphasising context representation differences.** To meet the objective of developing a new text classification dataset emphasising differences in context representation, it is essential to establish criteria for dataset creation that effectively capture the diverse contextual nuances present in real-world text data. However, existing datasets may not fully address the need for nuanced contextual variations, thereby limiting the effectiveness of algorithms trained on them. This particularly evident in question classification datasets, where question text is classified based on its possible or expected answer, presenting a multiclass classification problem. To further explore this area, this chapter will discuss literature related to question classification, focusing on methods and datasets to define the suitable criteria that best showcase the contextual differences in questions.

GAP 3: Evaluating the developed algorithm in the contextual text classification problem. Evaluating the developed modified word vectors algorithm (from Objective 1) in solving the contextual text classification problem necessitates benchmarking its performance against existing approaches. However, the literature lacks comprehensive studies assessing the effectiveness of algorithms that modify word vector weights in enhancing context understanding for text classification. Given this gap, this chapter aims to identify ways in which such evaluation can be performed, starting with fundamental literature of Machine Learning (ML) where classification problems have been a major area upon which the entire ML field is structured. Through this literature, the ML model, method, or algorithm best suited to this research's case, which involves multiclass classification of contextual text data, will be selected as the implementation approach. The developed ML system for the multiclass text classification problem will then be evaluated based on the baseline score value using various evaluation metrics.

Based on these three main areas, the following sections will sequentially explore the detailed literature, covering all required areas. Before summarizing this chapter, a *Direction of this Research* section will discuss the literature findings pertaining to these three main gap areas. These findings will then guide the structuring of the research methodology for the entire study.

2.3 Word Representations in Vector Space

Word Representation (WR), also known as word embedding or vector representation of words, refers to a technique that represents words as real numbers in a multidimensional vector space (Naseem et al., 2021; Jiao & Zhang, 2021; Patil et al., 2023, Incitti et al., 2023). The purpose of WR is to enable computers to interpret human language meaningfully by incorporating semantic metadata. Before WR, computers could only syntactically interpret words, based on factors such as capitalization and character length. For instance, the words "Morning," "Afternoon," and "Night" might be treated as three separate words with seven, nine, and five characters, respectively. However, with WR, these words can be represented using semantical metadata, such as their association with the category "Time of the day," which can be encoded as real numbers. As a result, computers can process the meaning of these words more accurately.

WR encodes the meaning of a word based on its relationship with other words. This method is originally derived from fundamental linguistic theories, such as the distributional hypothesis, which states that "Words that occur in the same contexts tend to have similar meanings" (Harris, 1954; Weaver, 1955), and the principle that "A word is characterised by the company it keeps" (Firth, 1957). Generally, WR represents a target word based on its relations with other words. For example, "Cat" might be represented as "Fur," "Meow," and "Pet"; "Dog" as "Fur," "Woof," and "Pet"; and "Car" as "Vehicle," "Move," and "Fast." With these representations, WR can interpret "Cat" and

"Dog" as semantically more similar than "Cat" and "Car," even though syntactically "Cat" and "Car" are more similar than "Cat" and "Dog."

Currently, there are two types of WR methods: Static (the older method) and Dynamic (the newer method). The Static method assigns a fixed vector to each word in the vocabulary (also known as the Word Embedding method) (Mikolov et al., 2013; Pennington et al., 2014), while the Dynamic method assigns an interchangeable vector to each word in the vocabulary (also known as the Contextualised Embedding method) (Devlin et al., 2018; Peters et al., 2018; Incitti et al., 2023). The Dynamic method was proposed to solve the polysemy problem faced by the Static/fixed representation. With the Dynamic method, polysemy can be solved using machine learning algorithms, as WR is calculated based on the currently processed sentence (Fodor et al., 2023). For instance, consider the word "Apple" in the sentences "I like eating Apple" and "I love Apple computers." These sentences provide distinct contexts for the word, with one referring to "Apple" as a type of fruit and the other as a brand name. Given the variability in meaning based on context, the representation of a specific word is contingent on the entire sentence provided as input. As a consequence of this requirement, a single word can be represented by an infinite number of embedding variations, given the limitless sentence variations in natural language. The achievement of contextualised word embeddings or vectors is facilitated through the implementation of Machine Learning (ML) algorithms, specifically computational statistical Language Models (LMs) (Asudani et al., 2023; Nandanwar & Choudhary, 2023; Worth, 2023).

In essence, a statistical Language Model (LM) functions as a probability distribution over a sequence of words, spanning phrases, sentences, paragraphs, and eventually entire text corpora. By incorporating this probability function into the parameters of ML, the model can dynamically calculate word embeddings during the inference process. To generate WR using ML, the algorithm doesn't store embeddings for individual words. Instead, it captures the patterns in which each word associates with others. Through extensive training on numerous natural language sentences, the algorithm learns and retains this pattern as its parameters (Elnagar et al., 2023). In live embedding calculation, the same word can possess different meanings based on its current context or sentence, effectively addressing the polysemy issue. In contrast, a precalculated embedding assigns the same meaning to a word unless the entire dictionary undergoes recalculation. Figures 2.1 and 2.2 visually depict this scenario.

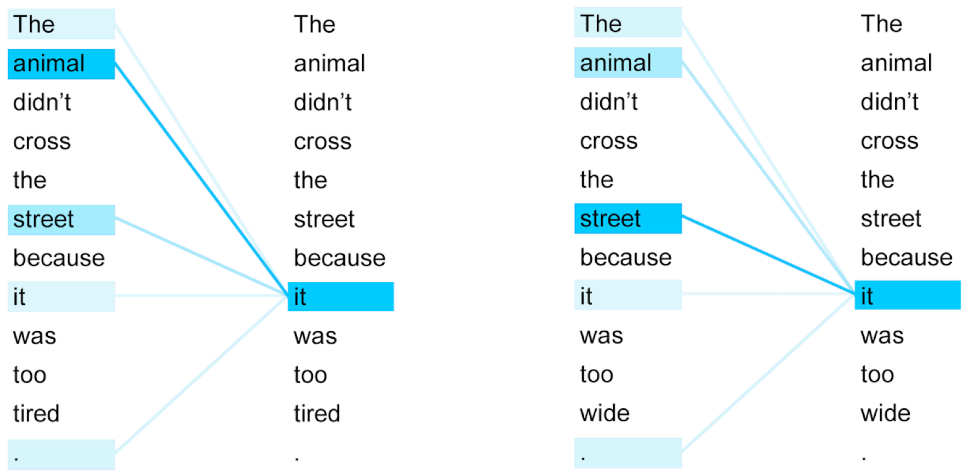

The animal didn't cross the street because it was too tired .
The animal didn't cross the street because it was too tired .
The animal didn't cross the street because it was too wide .
The animal didn't cross the street because it was too wide .

Figure 2.1 Transformer's self-attention mechanism (live calculation)
Source: Uszkoreit (2017).

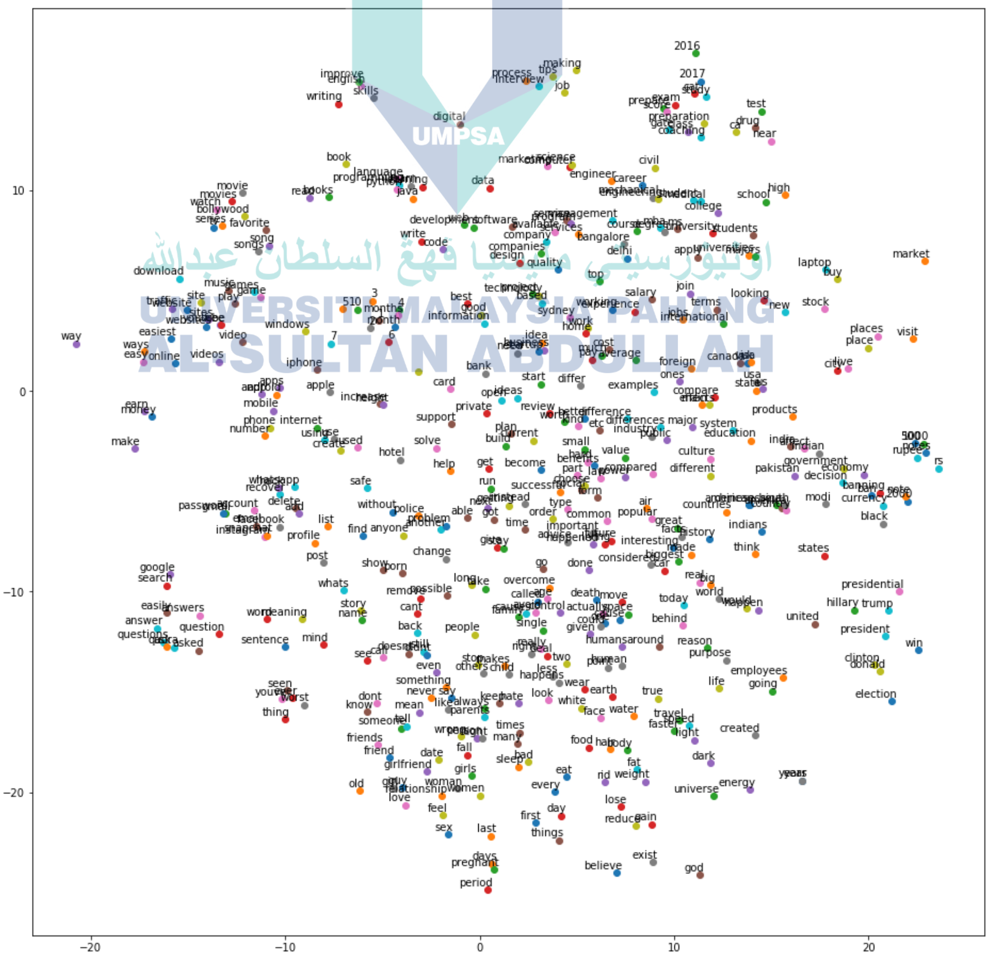


Figure 2.2 2D t-SNE projection of static word embedding (pre-calculated)

Figures 2.1 and 2.2 provide a visual comparison between live calculation and pre-calculated word representation/embeddings, respectively. In Figure 2.1, the illustration showcases the application of an attention mechanism, derived from the Transformer architecture (Vaswani et al., 2017), commonly employed in modern computational language models or Large Language Models (LLMs). This attention mechanism projects the semantic probability of each word towards other words in a given sentence. For example, the word "it" in the left sentence refers to "animal," while the same word "it" in the right sentence refers to "street," as depicted by the high contrast of the blue color. Despite being the same word, the differing contexts in the left and right sentences, such as "... it was too tired." and "... it was too wide.," lead the algorithm to calculate it differently, illustrating dynamic calculation. In contrast, Figure 2.2 illustrates all words in the given dictionary projected into a 2D embedding space using t-SNE algorithm (Arora et al., 2018). In this static calculation method, the probability distribution of all words in the dictionary is calculated against each other, resulting in the embedding for all those calculated words. Consequently, for this method, the word "it" will always have the same embedding unless the entire dictionary is recalculated with additional texts. This static approach contrasts with the dynamic nature of live calculation, highlighting how context-dependent meanings are captured in real-time during language processing tasks.

2.3.1 Word Weightage in Word Representation

In the process of word representation, the weighting of a word plays a crucial role in capturing its significance within a given context. This involves assigning numerical values, commonly referred to as weights, to individual words based on their contextual importance (Naseem et al., 2021). These weights signify the semantic relevance and influence of a word within the broader context of a sentence, paragraph, or document. Various approaches are employed to determine the weight of a word in word representation. Most of these approaches can be classified into two major categories: statistical encoding based on a word’s frequency and neural network encoding based on contextual relationship (Patil et al., 2023).

Statistical encoding methods, fundamental to word representation, analyse word frequencies within a corpus to assign numerical values, forming vectors that encapsulate the statistical distribution of words. The utility of these vector representations lies in their facilitation of linear algebra operations, allowing the manipulation of vectors, compute distances, and assess similarities. Example of such methods are Bag of Words (BOW) (Harris, 1954), N-gram (Katz, 1987), Term Frequency (TF) Embedding (Salton & Lesk, 1968), Hyperspace Analogue to Language (HAL) (Lund & Burgess, 1996), and Term Frequency-Inverse Document Frequency (TF-IDF) Embedding (Jones, 2004). While these methods effectively capture word distributions, they may face challenges in capturing intricate semantic relationships compared to more recent neural network-based approaches.

Neural network encoding methods represent a significant advancement in the field of word representation, particularly in capturing complex semantic relationships within natural language. These methods leverage the power of deep learning architectures to embed words into continuous vector spaces, allowing for a more dynamic and nuanced understanding of contextual meanings. One prominent neural encoding method is Word2Vec, introduced by Mikolov et al. (2013). Word2Vec operates on the principle of learning word embeddings by predicting a word's context in a given sentence. It captures relationships between words based on their co-occurrence patterns, producing embeddings that encapsulate semantic similarities. Global Vectors for Word Representation (GloVE), developed by Pennington et al. (2014), is another notable neural encoding method. GloVE combines both global statistics of word co-occurrence and local context window information to generate word embeddings. This method emphasises capturing not only direct relationships but also the global semantic structure within a corpus. Bidirectional Encoder Representations from Transformers (BERT), introduced by Devlin et al. (2018), represents a breakthrough in contextualised word embeddings. BERT employs a transformer architecture to consider the entire context of a word within a sentence, allowing for a more fine-grained representation that considers the specific ordering of words. These neural network encoding methods depart from fixed-length representations found in statistical methods. Instead, they adapt dynamically to the contextual nuances of language, enabling a more accurate portrayal of word meanings.

In summarizing this literature, Table 2.1 presents notable word encoding methods, categorising them into statistical and neural network types, and outlining their corresponding weight calculation methods. This table serves as a comprehensive overview of the evolution and variations within word encoding techniques over time, providing insights into the progression of methodologies in the NLP field.

Table 2.1 Notable word encoding methods

Method	Type	Weight calculation method
Bag of Words (BOW) (Harris, 1954)	Statistical	Frequency-based
N-gram (Katz, 1987)	Statistical	Frequency-based
Latent Semantic Analysis (LSA) (Evangelopoulos, 2013)	Statistical	Singular Value Decomposition (SVD)
Latent Dirichlet Allocation (LDA) (Blei et al., 2003)	Statistical	Probabilistic model, Generative
Hyperspace Analogue to Language (HAL) (Lund & Burgess, 1996)	Statistical	Context-based, Co-occurrence
Term Frequency (TF) Embedding (Salton & Lesk, 1968)	Statistical	Frequency-based
Term Frequency-Inverse Document Frequency (TF-IDF) Embedding (Jones, 2004)	Statistical	Frequency and Inverse Document Frequency
Word2Vec (Mikolov et al. 2013)	Neural	Context-based, Co-occurrence
Continuous Bag of Words (CBOW) (Mikolov et al. 2013)	Neural	Context-based, Co-occurrence
Skip-Gram (Mikolov et al. 2013)	Neural	Context-based, Co-occurrence
Doc2Vec (Le & Mikolov, 2014)	Neural	Context-based, Distributed Memory
Paragraph Vector (PV-DM) (Le & Mikolov, 2014)	Neural	Context-based, Distributed Memory
Paragraph Vector (PV-Dbow) (Le & Mikolov, 2014)	Neural	Context-based, Distributed Bag of Words
GloVE (Pennington et al. 2014)	Neural	Global statistics and local context window information
ELMo (Embeddings from Language Models) (Peters et al., 2018)	Neural	Contextualised embeddings, Bi-directional LSTM
FastText (Bojanowski et al., 2017)	Neural	Subword embeddings, CBOW
BERT (Devlin et al. 2018)	Neural	Contextualised embeddings, Transformer architecture
Universal Sentence Encoder (USE) (Cer et al., 2018)	Neural	Contextualised embeddings, Transformer architecture
GPT (Generative Pre-trained Transformer) (Radford et al., 2018)	Neural	Contextualised embeddings, Transformer architecture

Table 2.1 Continued

Method	Type	Weight calculation method
GPT-2 (Radford et al., 2019)	Neural	Contextualised embeddings, Transformer architecture
GPT-3 (Brown et al., 2020)	Neural	Contextualised embeddings, Transformer architecture
XLNet (Yang et al., 2019)	Neural	Contextualised embeddings, Transformer architecture
RoBERTa (Y. Liu et al., 2019)	Neural	Contextualised embeddings, Transformer architecture
ALBERT (Lan et al., 2019)	Neural	Contextualised embeddings, Transformer architecture
T5 (Raffel et al., 2019)	Neural	Text-to-text approach, Transformer architecture

From the listed methods in Table 2.1, the bottom nine methods, namely BERT, USE, GPT, GPT-2, GPT-3, XLNet, RoBERTa, ALBERT, and T5, can be classified as Large Language Models or LLMs. To further elaborate on this, the next section will discuss the literature of LLMs.

2.4 Large Language Model

Modern NLP programs typically consist of a specific type of computational Language Model (LM) called the Large Language Model (LLM). LLMs utilize deep learning techniques, specifically Neural Networks (NNs) with a large number of parameters in their architecture (also known as Deep Neural Language Models or DNLMs). In general, a word encoding/embedding method is considered to be an LLM when the trainable parameter size is large (the smallest LLM being BERT with 340 trainable parameters (Devlin et al., 2018)), is pre-trained using large corpora (billions of words or more), can demonstrate contextual understanding of natural language words (able to produce contextualised word vectors or embeddings), can be fine-tuned for specific NLP tasks, and historically implements the Transformer architecture (as demonstrated in previous Table 2.1). Despite its specific architecture, LLMs are still fundamentally a type of LM, a statistical model that learns patterns and relationships between words, phrases, and sentences in a given language. However, LLMs are trained on much larger datasets and use much more complex algorithms than traditional LMs, allowing them to perform more complex NLP tasks with greater accuracy.

Overall, the development of modern NLP systems consists of two main components: converting input text into numerical format (WR) and designing models to process this numerical data for solving practical problems in NLP. These practical problems are called downstream tasks, and contextualised embeddings play a crucial role in solving them. This claim is supported by the implementation of dynamic WR methods in many state-of-the-art LLMs for downstream tasks (Naseem et al., 2021; Singh & Mahmood, 2021; X. Liu et al., 2022; Incitti et al., 2023). Through publicly published leaderboards, some models even surpass human benchmarking performance. For example, the widely used SQuAD 2.0 QA dataset (Rajpurkar et al., 2018) currently has a state-of-the-art model that scored 93.214 in the F1 metric compared to 89.452 for human performance (https://rajpurkar.github.io/SQuAD-explorer/). Similarly, the CoQA QA dataset (Reddy et al., 2019) currently has a state-of-the-art model that scored 90.7 for overall F1 metric (Ju et al., 2019) compared to 88.8 for human performance (https://stanfordnlp.github.io/coqa/).

Apart from QA downstream tasks, these contextualised embedding LLMs have also achieved state-of-the-art in various other downstream tasks, such as Neural Machine Translation (NMT), Reading Comprehension, Text Summarisation, Common Sense Reasoning, Zero-Shot, Natural Language Inference (NLI), Sentiment Analysis, Co-reference Resolution, Document Classification, Sentence Classification, Semantic Textual Similarity, and Semantic Relevance. Table 2.2 lists some models that have achieved state-of-the-art results in various downstream NLP tasks.

Table 2.2 State-of-the-art LLMs in various downstream NLP tasks

Model name	**Downstream NLP task**
GPT- 1, 2 and 3	QA, NMT, Reading Comprehension, Text Summarisation, Common Sense Reasoning, Zero-Shot
XLNET	Reading Comprehension, NLI, Sentiment Analysis, QA
BERT	Sentence Classification, QA, NLI
RoBERTa	Sentiment Analysis, QA, NLI
ALBERT	Reading Comprehension, Semantic Textual Similarity, QA, Language Inference
T5/mT5	More diverse and challenging Coreference, Entailment, QA

Source: (Singh & Mahmood, 2021).

As mentioned in Chapter 1, Large Language Models (LLMs) are typically trained in two stages: *pre-training* for language understanding using large corpora, such as news articles and Wikipedia pages, and *fine-tuning* for specific downstream tasks, such as question answering and sentence classification, using task-specific datasets. During pre-training, the model is trained on a large amount of text to learn the general patterns and meanings of words in context. This process is usually unsupervised and focuses on optimizing the objective of predicting masked or next-word tokens based on their surrounding context. Pre-training can be done on the original model structure, such as the bidirectional language model used in BERT (Devlin et al., 2018), or on modified architectures, such as the encoder-decoder structure used in GPT-2 (Radford et al., 2019). Once the model has been pre-trained, it can be fine-tuned for specific downstream tasks by training on task-specific datasets with labelled examples. Fine-tuning requires modification and/or additional components to the pre-trained model, such as adding task-specific output layers or input embeddings. Fine-tuning is supervised and focuses on optimizing the objective of the downstream task, such as predicting the correct answer to a given question or classifying the sentiment of a sentence. Figure 2.3 provides an overview of the pre-training and fine-tuning process for LLMs, which results in a ready-to-use NLP model for downstream tasks.

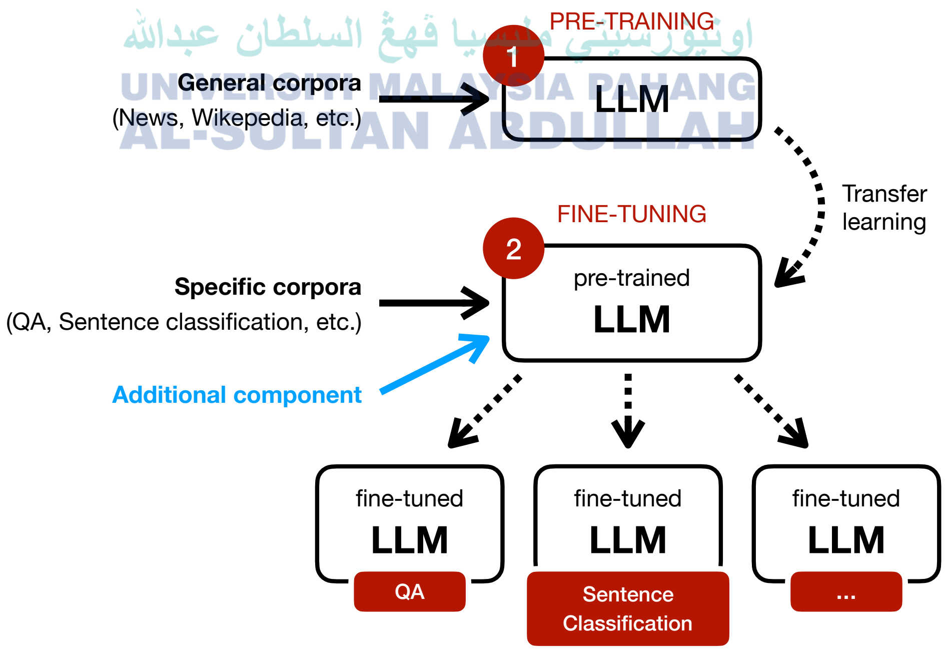


Figure 2.3 LLM transfer learning process

To better illustrate the pre-training and fine-tuning processes, let's consider the task of classifying English film reviews as either positive or negative. In this scenario, an LLM would be pre-trained using general English corpora (unsupervised pre-training on large datasets). After pre-training, a binary classification component would be added to the LLM, and the entire model would be retrained (with previously learned weights) on a dataset of English film reviews labelled as positive or negative (supervised fine-tuning using a smaller dataset). Once both the pre-training and fine-tuning processes are complete, this NLP model can be used to classify new film review comments as either positive or negative, effectively solving the task at hand.

While the transfer learning approach with pre-trained LLM is versatile, it can be expensive from an economic perspective. This is because the fine-tuning process requires the entire LLM to be trained on the task-specific dataset, even if the needed task is relatively simple, such as binary classification. LLM require an enormous number of parameters, ranging from millions to billions, to understand natural human language, which makes them computationally expensive to train. To further analyse this, the next subsection will discuss the literature surrounding the economics of LLMs.

2.4.1 The Economics of Large Language Models

The economics of LLMs are heavily influenced by their parameter size. Although the architectural complexity of each model may vary, the number of parameters is the main factor determining the amount of computing power required for training and inference. This claim is supported by Sharir's study, which calculated the cost (in USD) of model training based on parameter counts. Table 2.3 summarizes the reported costs for training the BERT models.

Table 2.3 The cost to train the different sizes of BERT models

Single run cost (in USD)	Fully loaded cost (in USD)	Parameter counts
2,500	50,000	110 million
10,000	200,000	340 million
80,000	1,600,000	1.5 billion

Source: Sharir et al. (2020).

Table 2.3 demonstrates the exponential increase in the cost of training BERT (Devlin et al., 2018) as the model size grows. The "Fully loaded" column in the table represents multiple training runs with hyperparameter tuning, which contributes to the overall cost of training the model. BERT is one of the most popular LLMs and is based on the Transformer architecture (Vaswani et al., 2017), which is a powerful machine-learning architecture for natural language processing tasks. Although LLMs based on the Transformer architecture can accumulate a large number of parameters, this architecture has been proven to be highly effective for various NLP tasks, leading to its widespread implementation in modern LLMs.

The next Table 2.4 lists several notable recent LLMs that are based on the Transformer architecture, sorted from smallest to largest in terms of parameter counts (which can also be translated as cheapest to most expensive in terms of training cost). The table includes information about the size (in terms of parameter counts), base architecture, and developer of each LLM. By examining the parameter counts and developers of these LLMs, insights into the economics of LLMs can be gained. Specifically, the development of LLMs is dominated by large tech companies such as Google, OpenAI, Microsoft, Nvidia and Facebook, which have the resources to fund the training and development of these large models. This suggests that the economics of LLMs are heavily influenced by the resources available to large tech companies and that smaller companies, institutes or individuals may struggle to compete in this space.

From Tables 2.3 and 2.4, it can be concluded that LLMs are extremely expensive to train. To provide a different perspective of the cost involved; it takes 4 days to train BERT (Large) (340 million parameters on 16 GB of training data) with 64 TPU chips (or approximately 1 day with 280 Tesla V100 GPU), 1 day to train Roberta (355 million parameters on 160 GB of training data) with 1024 Tesla V100 GPU, and 2.5 days to train XLNET (340 million parameters on 113 GB of training data) with 512 TPU chips. From a dollar perspective, training the BERT model with 16 GB data can cost anywhere between \$50k and \$1.6m (depending on the chosen model size and training procedure) and training the T5 model for a single run cost well above \$1.3m (Sharir et al., 2020).

Table 2.4 Notable recent LLM size, architecture and developer

Model name	Parameter counts	Base architecture	Developer
ELMo (Peters et al., 2018)	94 million	LSTM	AllenNLP
BERT (Large) (Devlin et al., 2018)	340 million	Transformer	Google
XLNET (Yang et al., 2019)	340 million	Transformer	Google Brain + CMU
RoBERTa (Y. Liu et al., 2019)	355 million	Transformer	Facebook
GPT-2 (Radford et al., 2019)	1.5 billion	Transformer	OpenAI
Megatron-lm (Shoeybi et al., 2019)	8.3 billion	Transformer	Nvidia
T5 (Raffel et al., 2020)	11 billion	Transformer	Google
Turing-NLG (Rosset, 2020)	17 billion	Transformer	Microsoft
GPT-3 (Brown et al., 2020)	175 billion	Transformer	OpenAI

As mentioned before, the current trend in emerging NLP models is versatility. LLMs transfer learning through pre-training and fine-tuning is currently the preferred method for solving downstream NLP tasks. While pre-training LLM requires significantly large computing resources, fine-tuning is more economically accessible as the required dataset for fine-tuning is much smaller than what is needed for pre-training (Megabyte for fine-tuning versus Gigabyte for pre-training). Nevertheless, fine-tuning is still relatively expensive due to the size of the chosen LLM. As previously mentioned, the fine-tuning process still needs to go through the whole LLM parameters despite the small dataset input and uncomplex pattern to be learned. Therefore, performing specific text classification (or other downstream tasks) through fine-tuning is still costly by model size metric and perhaps there are more economical approaches for this endeavour. To further narrow down the case study for this research, the question classification literature will be explored as the contextual text classification problem to be solved. The next section will further discuss the literature surrounding this area.

2.5 Contextual Text Classification

Contextual text classification is a dynamic approach to categorising text data based on its surrounding context and content. Unlike traditional text classification methos that rely solely on the text itself, contextual classification considers the broader context in which the text is situated, such as the user's intent, historical interactions, and the environment in which the text was generated. By leveraging modern word representation vectors through the LLM, contextual text classification systems can capture nuances in language, disambiguate meanings, and adapt to changing contexts in real-time (Dogra et al., 2022; Qassim et al., 2022; Bilal & Almazroi, 2023; Zhang et al., 2024). This enables more accurate and nuanced classification results, making contextual text classification invaluable for various applications such as sentiment analysis, topic modeling, and more. One domain that can greatly benefit from this approach is question classification. To further explore the literature surrounding contextual text classification, the following subsections will utilize question classification as a case study domain in which contextual text classification and dynamic word representation vectors can be implemented to further demonstrate its effectiveness and applicability across different contexts and domains.

2.5.1 Question Classification

Question classification in NLP is an area of study that focuses on categorising questions based on their intended meaning or purpose. It plays a fundamental role in various applications, including information retrieval, dialogue systems, and virtual assistants. The goal of question classification is to automatically assign a category, label or class to a given question, enabling systems to understand user queries and provide appropriate responses. Additionally, question classification encompasses different levels of granularity, ranging from simple binary classifications (e.g., yes/no questions) to more complex categorisations based on the type of information sought or the intent behind the question. For example, the question “Where is the Eiffel Tower located?” expects an answer such as “Paris,” “City of Paris,” or “France,” therefore the question classification system categorises this question under a “location” class. Similarly, “What is the capital city of Malaysia?” and “Where is the highest point in Japan?” also fall under the “location” class based on their expected answers.

In practical applications, question classification methods (i.e., question classifiers) are often implemented in Question-Answering (QA) and Information Retrieval (IR) systems. To retrieve the correct answer, the QA system needs to know what to look for in the given context. With the output from the question classifier, the QA system can eliminate the need to search for unrelated contexts, reducing the processing time (e.g., by only searching for "location" related information instead of all possible information). However, this type of question classifier has become less relevant with current QA system methods. To elaborate further on this, the next subsection will discuss the literature surrounding question classification methods, their related datasets, and the current landscape about the relevance of such methods.

2.5.2 Question Classification Methods and Datasets

Question classification methods aim to semantically classify questions based on a defined taxonomy (Gupta et al., 2021). One of the most widely used question taxonomies to date is from the work of Li and Roth (Li & Roth, 2002) known as The Text Retrieval Conference (TREC) dataset. This dataset includes a total of six coarse classes and 50 fine classes, which are listed in the following Table 2.5.

Table 2.5 Coarse and fine classes for 500 questions in the TREC dataset

Coarse class	**Associated fine class**
ABBREV.	abb, exp
ENTITY	animal, body, colour, creative, currency, dis.med., event, food, instrument, lang, letter, other, plant, product, religion, sport, substance, technique, term, vehicle, word
DESCRIPTION	definition, description, manner, reason
HUMAN	group, individual, title, description
LOCATION	city, country, mountain, other, state
NUMERIC	code, count, date, distance, money, order, other, period, per cent, speed, temp, size, weight

Source: Li & Roth (2002).

Table 2.5 provides a list of question taxonomy classes for 500 sampled questions from the TREC dataset (Li & Roth, 2002). This dataset has been publicly available and has been used by many researchers for developing various question classification (and other sentence classification) methods. Recent studies that used the TREC dataset include Zhong et al.'s GPT-3 based method for classifying sentence descriptions based on taxonomy (Zhong et al., 2022), Chen et al.'s Dual Contrastive Learning (DualCL) method for classifying sentences through label-aware data augmentation (Chen et al., 2022), Jardim et al.'s CNN method for classifying questions in Portuguese (Jardim et al., 2022), Hamza et al.'s TF-IDF weighting method for classifying Arabic language question text (Hamza et al., 2021), Chotirat and Meesad's method based on Part-of-Speech (POS) for question classification of Thai language wh-question text (Chotirat & Meesad, 2021), Sangodiah et al.'s Term Weighting method (E-TFIDF and TFPOS-IDF) for classifying exam questions (Sangodiah et al., 2021), Golzari et al.'s question classification method based on Differential Evolution (DE) and Gravitational Search Algorithm (GSA) (Golzari et al., 2022), and Weerakoon and Ranathunga's fine-tuned BERT model for question classification in the travel domain (Weerakoon & Ranathunga, 2021).

Although still an active research area, methods for taxonomy-based question classification for QA systems have become increasingly irrelevant in recent years. This is mainly due to advancements in modern QA systems that have made the taxonomy identification process for question text classification an unnecessary function. As mentioned in subsection 2.4.1 *QA System Methods and Datasets*, modern QA systems are capable of learning the semantic relationship between QA pair text directly within their core Language Model (LM) architecture (through the fine-tuning process) (Ju et al., 2019; Yamada et al., 2020; Zhang et al., 2021). With these types of LM embedded in QA systems, identifying 'location' as the intent of the "Where is the Eiffel Tower located?" question has become an automated process without the need for an additional classifier. To provide further evidence for this argument, Table 2.6 compares the coarse classes from Li and Roth's taxonomy with the answer types from benchmark QA datasets as defined by Yatskar (2019).

Table 2.6 Taxonomy class against answer types from QA datasets

Li and Roth Coarse class	Yatskar answer type
ABBREV.	Extractive or Abstractive answer (Fluency/ Coreference)
ENTITY	Extractive answer
DESCRIPTION	Extractive or Abstractive answer (Fluency/ Coreference)
HUMAN	Extractive answer
LOCATION	Extractive answer
NUMERIC	Abstractive answer (Counting)
Not available	Unanswerable
Not available	Abstractive answer (Yes/No and Picking)

Table 2.6 presents a comparison between Li and Roth's question taxonomy coarse class with Yatskar's answer types for SQuAD 2.0, CoQA, and QuAC datasets. The comparison shows that three taxonomy classes correspond to extractive answers, two classes correspond to extractive or abstractive answers (*Fluency* or *Coreference*) depending on the context, and one taxonomy class corresponds to abstractive answer (*Counting*). However, two of Yatskar's answer types are missing from Li and Roth's question taxonomy course class, namely *Unanswerable* questions and abstractive answers related to *Yes/No* and *Picking* phenomena. This comparison further justifies that taxonomy classes for modern QA systems are not required as most of them are for extractive-type answers. However, modern QA systems do not cater to question text classification based on question and general word occurrences (referred to as Question Type Classification in this document). Figure 2.4 depicts a scenario where this type of classification is needed to assist the QA system in handling abstractive-type answers.

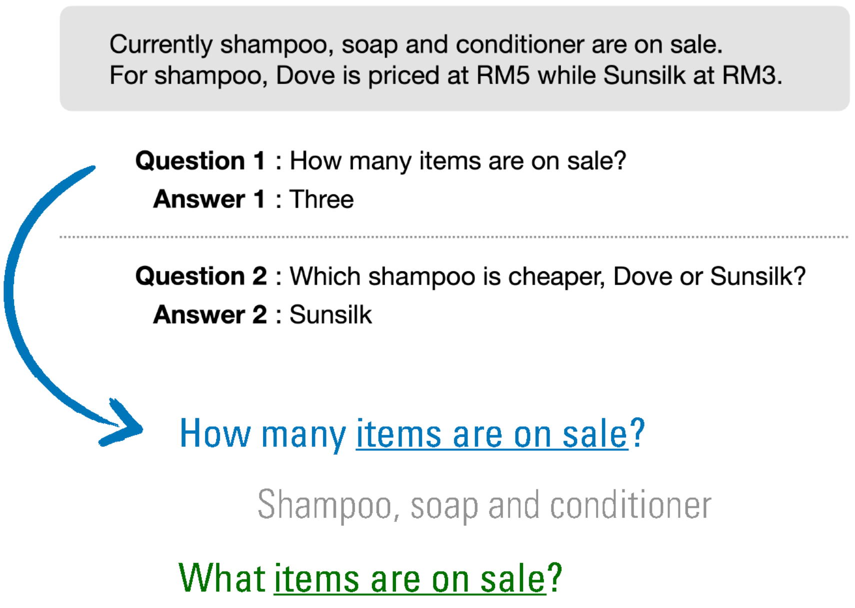

Currently shampoo, soap and conditioner are on sale.
For shampoo, Dove is priced at RM5 while Sunsilk at RM3.
Question 1 : How many items are on sale?
Answer 1 : Three
Question 2 : Which shampoo is cheaper, Dove or Sunsilk?
Answer 2 : Sunsilk
How many items are on sale?
Shampoo, soap and conditioner
What items are on sale?

Figure 2.4 Question word role in question classification

Figure 2.4 illustrates a scenario where a question-type classifier based on question words is necessary to assist the QA system. The context in this simplified example is "Currently shampoo, soap and conditioner are on sale.…" and the question is "How many items are on sale?" Using a modern QA system, the keywords extracted from the question text would be "items are on sale," leading the system to retrieve "Shampoo, soap and conditioner are on sale" as the answer. However, this answer is not the correct answer, instead, should be "Three" (a counting phenomenon under the abstractive answer feature). On the other hand, the retrieved answer from the system could be the answer to the question "What items are on sale?". The difference between the two questions lies in the question word "How many" and "What," not in the keywords/function words. To address this issue, it is desirable to have a classifier that can classify question text that focus on both extractive type answers and abstractive answers phenomenon. This problem can be classified as a multiclass classification problem, where the input (question text) needs to be classified into one of multiple classes (one of abstractive answer phenomena or extractive type answer).

Solving this type of multiclass classification problem can be challenging, particularly due to the requirement that the classifier must be able to capture the contextual semantics of the question. With such complexity, there are currently no methods that surpass machine learning in dealing with such intricacies. Moving forward, the following section will explore the machine learning literature, focusing on multiclass classification of natural language text data.

2.6 Machine Learning

Machine Learning (ML) refers to a computerised system that automatically learns patterns from data. Unlike traditional computer systems, where logic is manually encoded into the algorithm, ML systems learn from patterns in the given training data and encode the logic into the algorithm automatically. This process allows ML systems to grasp complex underlying patterns that are impossible to be manually encoded by human programmers. There are three ways to 'teach' the ML algorithms: *Supervised*, *Unsupervised*, and *Reinforcement* learning. *Supervised* learning involves the algorithm learning from labelled examples, while *Unsupervised* learning involves the algorithm discovering patterns in the data on its own. *Reinforcement* learning, on the other hand, learns by following a guideline that provides rewards or punishments. Figure 2.5 illustrates the most common ML algorithms under *Supervised* and *Unsupervised* learning. It's worth noting that *Reinforcement* learning is not included as it is in a far-off domain from this research's aim.

As shown in Figure 2.5, ML algorithms can be further categorised into three subcategories: *Classification*, *Regression*, and *Clustering*. These subcategories correspond to the types of problems that these algorithms are capable of solving. In *Classification*, the algorithm is tasked with classifying the input into one of two classes/labels (binary classification) or more than two classes/labels (multiclass classification). An example of a classification problem is identifying whether an input image is a cat or a dog. In *Regression*, the algorithm needs to predict continuous output based on given input features. An example of a regression problem is predicting the value of a house based on features such as year of construction, location, and size. In *Clustering*, the algorithm learns to categorize input data into a certain number of groups based on input features. An example of a clustering problem is dividing a group of people into two subgroups based on their family, education, and work background. Figure 2.6 visualizes the concepts of classification, regression, and clustering in their simplest forms.

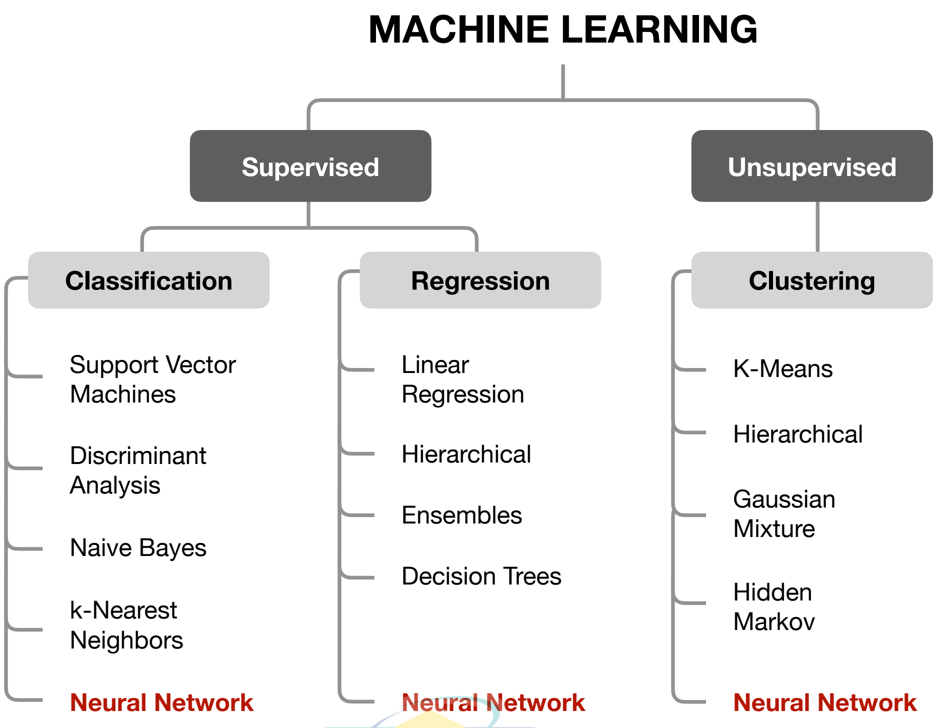


Figure 2.5 Classification of the most common machine learning algorithms Source: Duc et al. (2019).

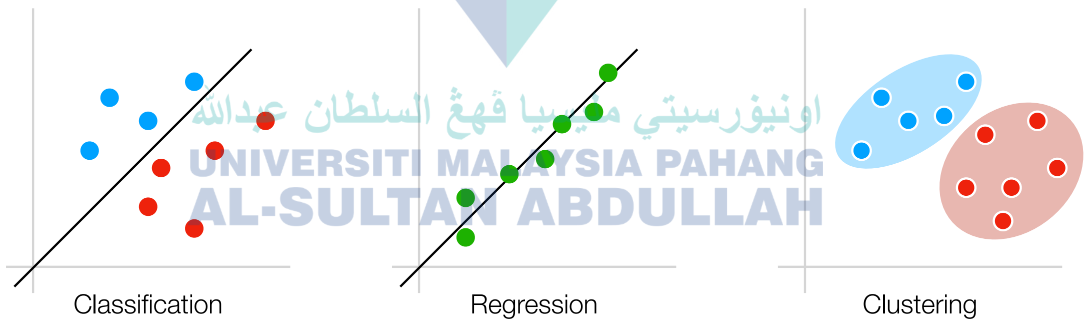


Figure 2.6 Simple visualization of Classification, Regression and Clustering

Referring back to Figure 2.5, one of the most versatile ML algorithms is the Neural Network (NN). It is capable of solving all three types of problems (classification, regression, and clustering), making it a dominant algorithm in the field. Unlike other algorithms that are relatively rigid and unable to be expanded, NN is highly flexible. This flexibility is demonstrated by its ability to be used as the basis for many other algorithms with different capabilities and architectures. Following Figure 2.7 highlights various NN algorithms that fall under Supervised, Unsupervised, and a combination of both learning methods.

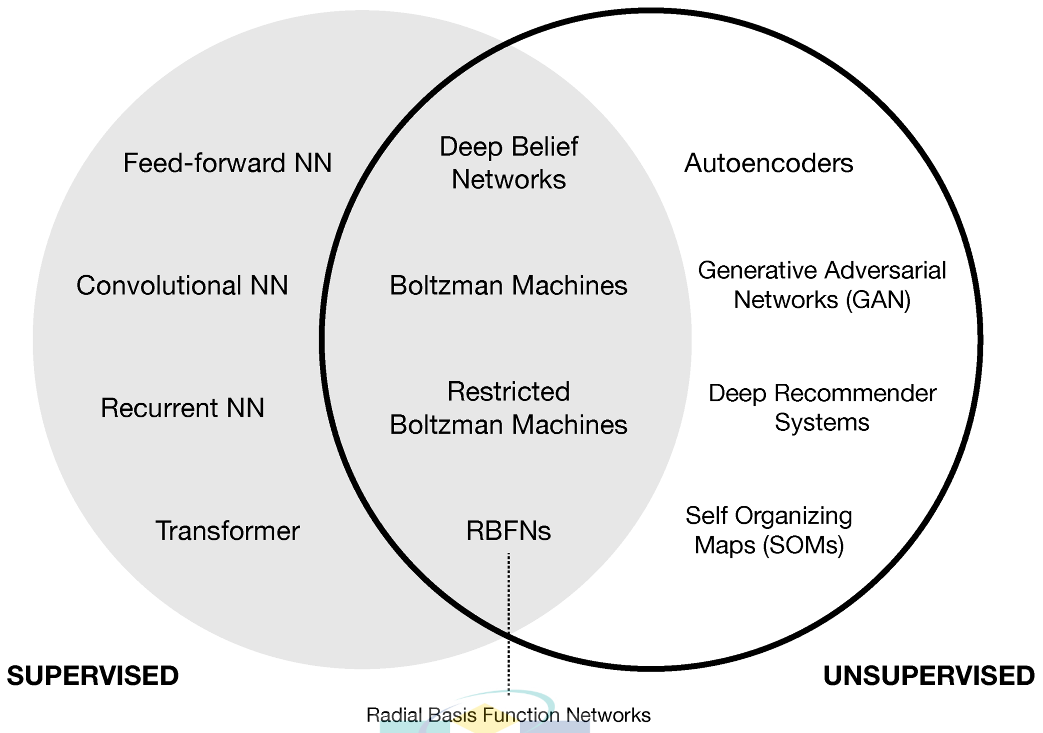


Figure 2.7 Supervised and unsupervised neural network models

To recap, the case study of this research is to solve a multiclass classification problem. Therefore, the following discussion will focus solely on ML algorithms within this domain. As shown in previous Figure 2.5, five ML algorithms can be used for classification: Support Vector Machines (SVM), Discriminant Analysis (DA), Naïve Bayes (NB), k-Nearest Neighbour (kNN), Neural Network (NN). In recent years, SVM and NN have been the most widely used algorithms, as evidenced by the number of publications between 2019 and 2024 found through a Google Scholar search. Table 2.7 provides an overview of this finding, although it should be noted that NN may have even more usage due to the various names under which NN implementations can be found, as shown in previous Figure 2.7.

Table 2.7 Google Scholar search results for publication year 2019 to 2024

Search keywords	**Result counts**
support vector machine classification	278,000
neural network classification	347,000
k-nearest neighbour classification	17,300
naive bayes classification	18,200
discriminant analysis classification	16,100

Due to their relatively low usage in recent years compared to SVM and NN, DA, NB, and kNN algorithms will not be discussed further in this section. Instead, the focus will be on SVM and NN algorithms, which are both capable of solving multiclass classification problems. As discussed in Chapter 1, this research aims to address an issue in downstream NLP tasks through multiclass classification. Table 2.8 provides further evidence for the suitability of SVM and NN algorithms for this task, based on Google Scholar search results.

Table 2.8 Google Scholar search results for publication year 2019 to 2024

Search keywords	Result counts
support vector machine multiclass classification	17,900
neural network multiclass classification	17,000

Although both SVM and NN can perform multiclass classification, they do so in different ways. SVM is a binary classifier, so to perform multiclass classification, it breaks down the problem into several binary classification problems, with one binary classifier per pair of classes. This approach leads to shorter training time but longer inference time due to the hierarchy of classification processes. In contrast, multiclass classification is natively supported by NN. By making the number of classes as NN output nodes, multiclass classification can be performed using the softmax function. The softmax function produces a sum of 1 output, and the highest value denotes the selected class. While NN takes longer to train, as each data point is compared to all classes in every iteration, it takes less time for inference compared to SVM, which requires multiple binary classifiers to run the whole classification process multiple times.

As previously mentioned, the case study of this research is to solve a multiclass classification problem on high-dimensional text data (word representation vectors or word embeddings). Therefore, the NN algorithm is selected for this research, as it is known to be highly effective in discovering intricate structures within high-dimensional data, such as vector data that contains contextual text information (LeCun et al., 2015). This is also the reason why modern LLMs use Transformer-based ML models, which are largely based on NN algorithms. With NN as the selected algorithm, further literature will focus on understanding this algorithm in an attempt to use it to design a multiclass classifier with a small parameter count.

2.6.1 Artificial Neural Network

An Artificial Neural Network (ANN/NN) is a computational model that is loosely inspired by a biological neuron (Lippmann, 1994). A single artificial neuron (unit) can receive multiple scalar values as its input and produce a single scalar value as an output. Figure 2.8 visualizes one unit of an artificial neuron, where $\{x_1, \ldots, x_n\}$ are inputs, $\{w_1, \ldots, w_n\}$ are weights, b is biased, and y is output (all values are in scalar unit). Each artificial neuron also consists of one activation function (also called link/decision/transfer function) denoted as f. Depending on the desired output, the activation function can be Sigmoid, Tanh, ReLU, and so on. Equation 2.1 represents an artificial neuron in mathematical form.

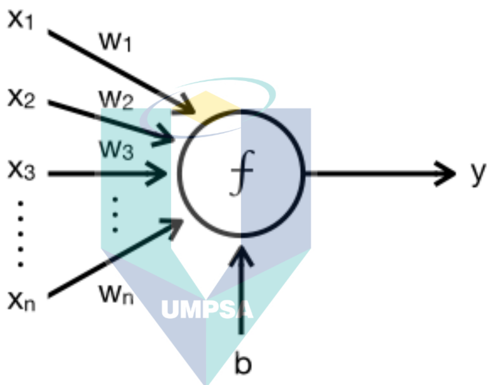


Figure 2.8 One unit of artificial neuron

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

From a single artificial neuron, it is possible to create an expandable system by interconnecting multiple artificial neurons through a series of layers, forming an *Artificial Neural Network* (ANN) or simply *Neural Network* (NN). The most basic construction of a NN consists of three layers: the *Input* layer, the *Hidden* layer, and the *Output* layer. Figure 2.9 visualizes this basic three-layer NN model where *x* represents the input neuron, *h* represents the hidden neuron, and *y* represents the output neuron. In terms of connectivity, all neurons in the prior layer are connected to all neurons in the subsequent layer, forming a *Dense* or *Fully Connected* layer. This form of NN is also known as a vanilla form of NN (NN's most basic form). In addition to the basic three layers, the hidden layers of an NN can be indefinitely expanded, as illustrated in Figure 2.10.

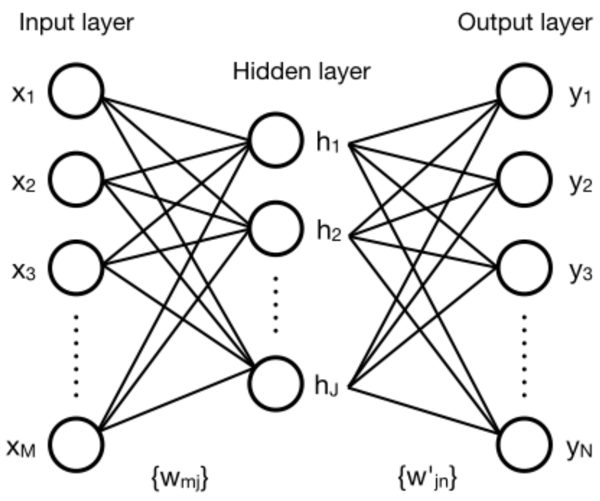


Figure 2.9 Basic/shallow Artificial Neural Network model (one hidden layer)

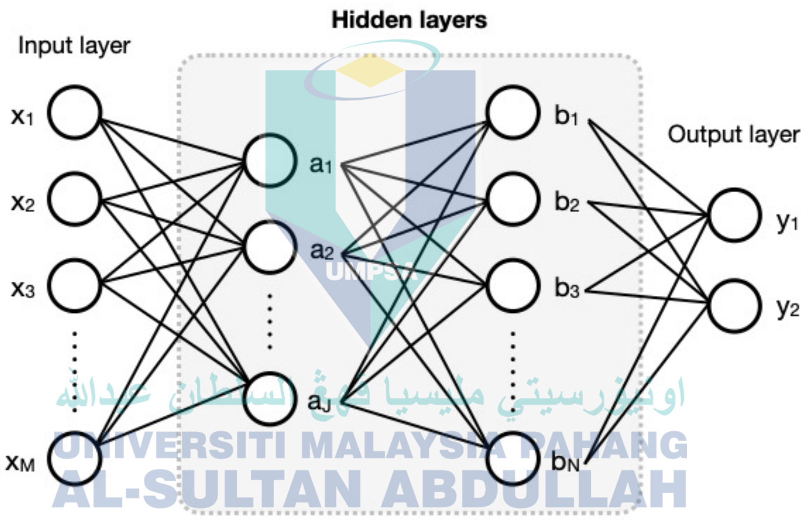


Figure 2.10 Deep Artificial Neural Network model (multiple hidden layers)

Previous Figures 2.9 and 2.10 showcase two forms of vanilla NN: Shallow NN (three layers) and Deep NN (more than three layers). In addition to these vanilla forms, a vast number of other NN models have been proposed over the years. As previously illustrated in Figure 2.7, these models are an expansion of vanilla NN with additional components, such as additional specific nodes/layers or specific mechanisms. These models, mainly of the Deep NN type, can be utilised using the previously discussed two ML training methods: *Supervised* and *Unsupervised*. To design a machine learning classifier to solve a multiclass classification problem, the next subsection will further discuss NN models under the Supervised category (the training method for solving classification problems as shown in the previous Figure 2.5).

2.6.2 Supervised NN

Like other supervised ML algorithms, supervised NN can be used to solve *Classification* and *Regression* problems. In general, a supervised NN Classification model will predict discrete values/classes/labels, while a supervised NN Regression model will predict continuous values/quantities. For example, consider a weather prediction ML model. The question to be answered for the Classification model is “Will it be cold or hot?”, while for the Regression model, it is “What will be the temperature?”. To visualize this example, the following Figure 2.11 illustrates the graph for Classification and Regression results.

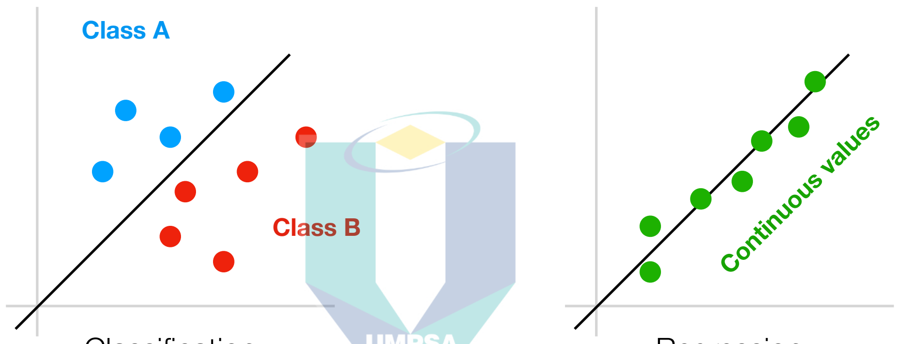


Figure 2.11 Classification versus Regression

As illustrated in Figure 2.11, the Classification model will predict whether it's going to be cold (Class A: blue-coloured dots) or hot (Class B: red-coloured dots), while the Regression model will predict the temperature value (continuous temperature values - green dots). For both models, the prediction will be made towards the black line in the graph. For Classification, the black line will decide whether it's cold or hot, while for Regression, the temperature will be predicted as a value as close as possible to the black line. This black line symbolizes what the model has learned after going through the training process, which involves analysing the given input and output data repeatedly. Note that in practice, the line would not be as linear as in the given figure, and the dimensions could also be more than two, depending on the dataset.

For classification, there are two types of problems: *Binary* and *Multiclass*. The previous example of predicting cold or hot weather is a Binary classification problem (predicting one from two classes). If the number of classes is more than two, it is called a Multiclass classification problem. An example of a Multiclass classification problem is predicting a student's grade in a computer science subject, where the grade can be from A+ to F (predicting one from more than two classes). Referring back to Figure 2.7, all the listed NN-supervised models can be used to solve both Binary and Multiclass classification problems. To recap, the four supervised-only NN models are Feed-forward NN, Recurrent NN, Convolutional NN and Transformer.

From an architectural perspective, the three bottom models (Convolutional NN, Recurrent NN, and Transformer) are extensions of the Feed-forward NN. Starting with the base/vanilla deep Feed-forward NN, these three models then implement several additional components to suit each respective architectural purpose. Figure 2.12 illustrates the general differences between the Feed-forward, Recurrent, and Convolutional NN model architecture (the Transformer will be illustrated later as its architecture is vastly different and more complex than the others).

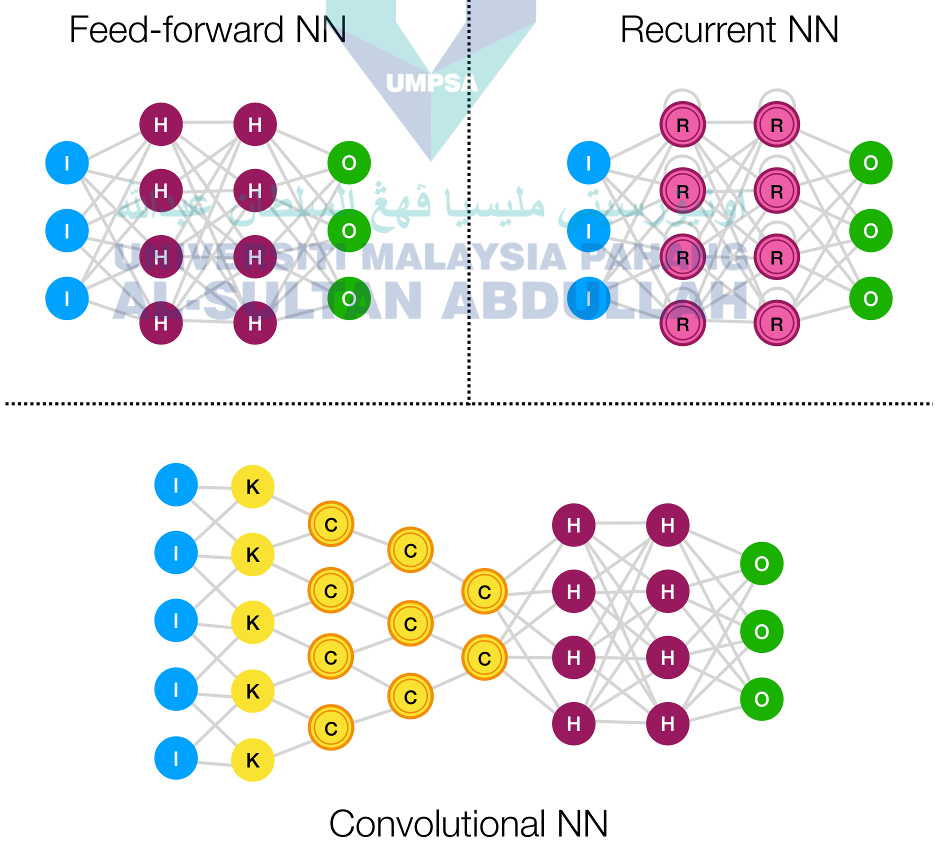


Figure 2.12 Feed-forward, Recurrent and Convolutional NN model architecture

Referring to Figure 2.12, the vanilla Feed-forward architecture (with input-hidden-output layers denoted by ‘I’, ‘H’, and ‘O’) can be seen implemented in both Recurrent and Convolutional NN. In Recurrent NN, the additional component added is the recurrent mechanism, which allows the hidden layer's nodes to calculate output based on both the current and previous inputs using the *Hidden State* mechanism (unlike vanilla Feed-forward NN, which only considers the current input) In the given figure, these nodes are denoted by the circled ‘R’ (Recurrent) with a looped link to itself, representing the recurrent calculation/mechanism. Recurrent NN is suitable for sequential data, and there are three commonly used variants of Recurrent NN: Vanilla RNN, LSTM (Long-Short Term Memory), and GRU (Gated Recurrent Unit). While vanilla RNN can only remember recent previous inputs in its hidden state, LSTM and GRU have an additional component called *Memory Cell*, which enables them to remember not only the recent previous inputs but also all previous inputs starting from the first item in the input sequence.

While Recurrent NN transforms vanilla hidden layers into recurrent hidden layers, Convolutional NN adds more layers while maintaining the vanilla hidden layers. Denoted as ‘K’ and circled ‘C’ in Figure 2.12, these additional layers contain nodes that are known as the *Kernel* and *Convolution* nodes. From input nodes, kernel nodes filter the input into a two-dimensional weight array before passing the value into convolutional nodes. Within convolutional nodes, the two-dimensional weight array is multiplied several times until just two values remain. This process narrows the dimension of the original input, allowing the model to focus on a certain input. The two convoluted values then pass to dense vanilla NN hidden layers for regular NN processing. As per its intended design, Convolutional NN is very good at classifying image data. This is because Convolutional NN focuses on certain image areas (multiple times) that are ultimately the key points in classifying the image, such as whether it is a ‘Dog’ or a ‘Cat’ image. Although Recurrent and Convolutional NN added more components into vanilla/Feed-forward NN, the main overall model is relatively similar. The Transformer model, on the other hand, is vastly bigger and different. The following Figure 2.13 illustrates the Transformer model architecture.

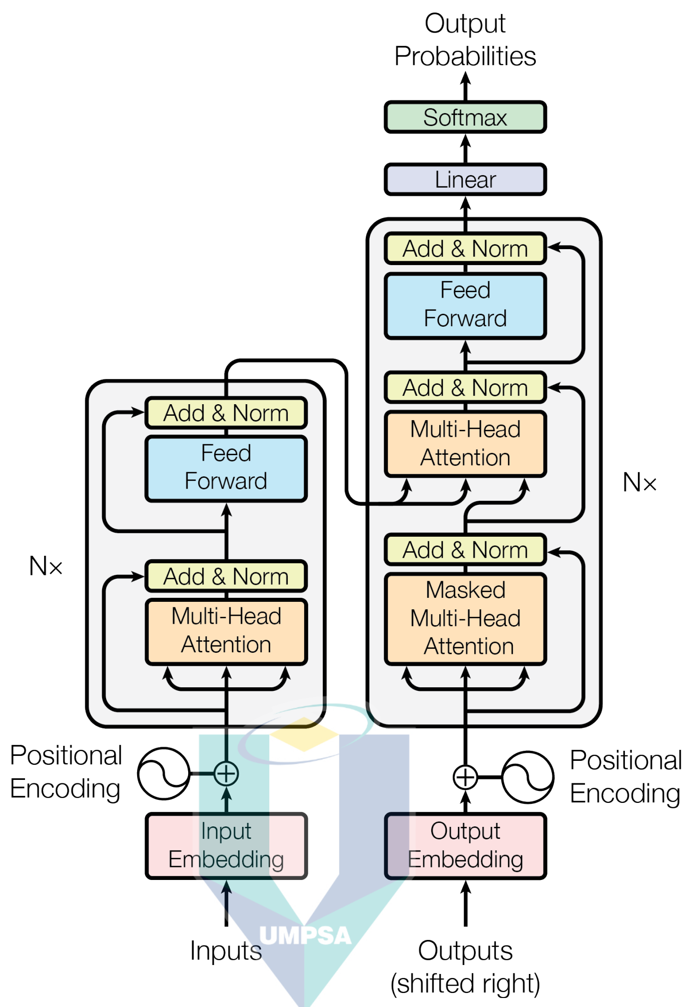


Figure 2.13 Transformer model architecture
Source: Vaswani et al. (2017).

Referring to Figure 2.13, two Feed-forward NN models can be seen being implemented in the Transformer model. Coloured blue in the figure, the two implementations shown are not two separate counts of Feed-forward models, but rather two locations or segments in which a Feed-forward NN is being implemented in the Transformer (one in the *Encoder* block on the left side of the Transformer and one in the *Decoder* block on the right side of the Transformer). The number of Feed-forward models in each implementation location depends on the implemented Transformer's input size. For example, BERT Language Model (Devlin et al., 2018) uses a 512 input size for its Encoder-only Transformer implementation, and therefore, there are 512 Feed-forward models in one BERT's Encoder block. It is important to note that BERT uses 12 blocks for its *base* model and 24 blocks for its *large* model. Therefore, for the BERT base model, there will be a total of 6144 Feed-forward models in the whole architecture (512 Feed-forward models in each of the 12 Encoder blocks).

Referring to the previously discussed NN models, namely Recurrent NN, Convolutional NN, Transformer, and Feed-forward NN, it is important to note that all of these models incorporated a Feed-forward NN component. As the original and simplest form of NN architecture, the Feed-forward is also the most economical in terms of the number of components and parameters compared to the other models. Table 2.9 below provides a comparison of the four supervised NN models and their implemented components.

Table 2.9 Supervised NN model's components comparison

	Components
Feed-forward	Input, hidden and output layers (*vanilla*)
Recurrent	- *Vanilla* + recurrent mechanism on hidden layers (base RNN) - *Vanilla* + recurrent + memory cell (LSTM) - *Vanilla* + recurrent + different memory cell (GRU)
Convolutional	*Vanilla* + kernel + convolutional cell
Transformer	*Vanilla* (multiple) + positional encoding + attention mechanism

With this literature on ML, this chapter is concluding its literature sections, and to concatenate and align all findings throughout the discussed topics, the next section will discuss the direction of this research.

2.7 Direction of this Research

The previous sections aimed to analyse and explore the current literature concerning the identified research gap, which surround three main areas defined in the gap analysis section. The following are the summarised findings for each gap area:

GAP 1: Incorporating external scalar weights into word representation vectors. From the discussed literature, it is evident that contextualised word representation vectors can only be generated using LLMs. Additionally, it is apparent that the word vectors generated by LLMs cannot be externally modified by incorporating external values. This limitation arises from the design of such models, which utilize a Transformer architecture consisting of multiple stacked encoder blocks. This complex structure results in the word vectors' trainable parameters being highly interconnected and dependent on each other. Given this situation, a new method for modifying the high-dimensional contextualised word vectors needs to be proposed to address this gap. For generating the word vectors in this research, the Universal Sentence Encoder (USE) (Cer et al., 2018) has been selected. This choice is based on two rationales:

1. USE is the only LLM that can generate small-sized word embeddings (100 dimensions), while the smallest embedding size for other LLMs is 512 dimensions. This embedding size is crucial in this research as it attempts to minimize costs as opposed to fine-tuning LLMs, which are high in cost.

2. As per its name, USE by default is an LLM designed to encode sentences (not words) into embeddings. Given that this research focuses on the question classification problem, where questions are essentially sentences comprised of multiple words, comparing the developed algorithm's performance using the original USE sentence embedding against modified word embeddings could offer a more insightful evaluation of the modified embedding vector's effectiveness (raw sentence embedding compared to modified word embeddings for generating the sentence embedding).

GAP 2: Defining criteria for creating text classification datasets emphasising context representation differences. When establishing criteria for the new contextual text classification dataset, the *Question Classification* research area has been chosen, particularly due to its intricacies in defining the question context not only within its own words but also by considering its possible or expected answer. Literature in this area has shown that current taxonomy-based datasets have become increasingly irrelevant in recent years due to advancements in modern QA systems based on fine-tuned LLMs. With this in mind, this research intends to use QA datasets as a benchmark for creating a new dataset for question classification, where the main goal is to identify whether the question expects an extractive-type answer or one of various abstractive-type answers. In contrast to current taxonomy-based question classification datasets, this new proposed dataset will focus on emphasising differences in the context or type represented by the questions, rather than creating detailed coarse and fine classes that can become very complex and numerous. With this focus, the question classification classes can be low in number and only require one level instead of two levels (coarse and fine) for the current taxonomy-based classes.

GAP 3: Evaluating the developed algorithm in the contextual text classification problem. To evaluate the algorithm developed to address the first identified gap, the literature review in this area explores text classification methods, with a specific focus on multiclass classification techniques based on machine learning as a benchmark for resolving high-dimensional data classification challenges. The ability to handle high-dimensional data is crucial due to the embedding size of LLM-generated word representation vectors. According to the literature, the vanilla feed-forward Neural Network (FF-NN) emerges as the optimal choice for this research. This selection is driven by two key factors: its cost-effectiveness and suitability for multiclass classification of high-dimensional data. Among the four NN architectures considered; vanilla Feed-forward, Convolutional, Recurrent, and Transformer, vanilla Feed-forward is identified as the most economical option, attributed to its simplicity and minimal number of components. Figure 2.14 illustrates the literature's path towards selecting the Vanilla Feed-forward NN as the ML model for this research.

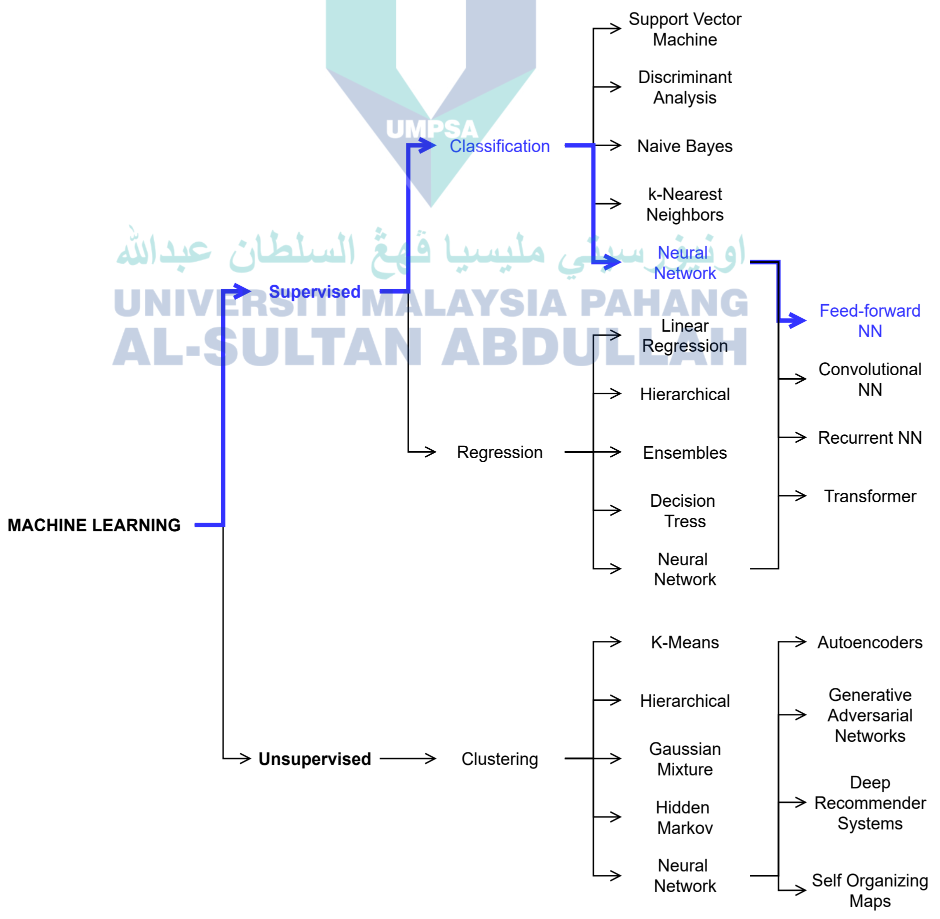


Figure 2.14 Literature path towards selecting the vanilla Feed-forward Neural Network (FF-NN) for this research

2.8 Summary

This chapter presents a literature review surrounding contextual text classification, aiming to gather information to address the three defined research questions: 1) How can an algorithm effectively incorporate external scalar weights into word representation vectors to enhance context understanding in a contextual text classification problem?, 2) What criteria should be considered in the creation of a text classification dataset to emphasise differences in context representation? and 3) How does the developed algorithm, incorporating external scalar weights, perform when applied to contextual text classification task? Three gap areas are defined for this research and based on the literature, several findings are outlined and discussed in previous *Direction of this Research* section. With this defined research direction, the next chapter will discuss the structured research methodology aimed at justifying the defined hypotheses for this entire study, ultimately closing the identified gaps and achieving the stated objectives while answering the research questions.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This study aims to justify the hypothesis that the word representation vector derived from the LLM can be altered using an external scalar weight, which can later be used as input for a ML model to perform text classification task. To test this hypothesis, three research objectives and their corresponding research questions were outlined in Chapter 1. To recap, the objectives and research questions are as follows:

1. To develop a new algorithm that incorporate an external scalar weight into the word representation vector. Research Question 1 (RQ1): How can an algorithm effectively incorporate external scalar weights into word representation vectors to enhance context understanding in a contextual text classification problem?

2. To develop a new text classification dataset that emphasises differences in context representation. Research Question 2 (RQ2): What criteria should be considered in the creation of a text classification dataset to emphasise differences in context representation?

3. To evaluate the developed algorithm using a Machine Learning (ML) model in the contextual text classification problem. Research Question 3 (RQ3): How does the developed algorithm, incorporating external scalar weights, perform when applied to contextual text classification task?

Based on these objectives, a research methodology consisting of five main phases is structured. The next section 3.2 will elaborate on this research methodology in detail.

3.2 Research Methodology

A five-phase methodology is structured for this research, comprising 1) Base Data Preparation, 2) Dataset Development, 3) Modified Word Vectors Method Development, 4) Machine Learning Classifier Development, and 5) Methods Evaluation. Following Figure 3.1 visualizes this methodology in a flowchart format.

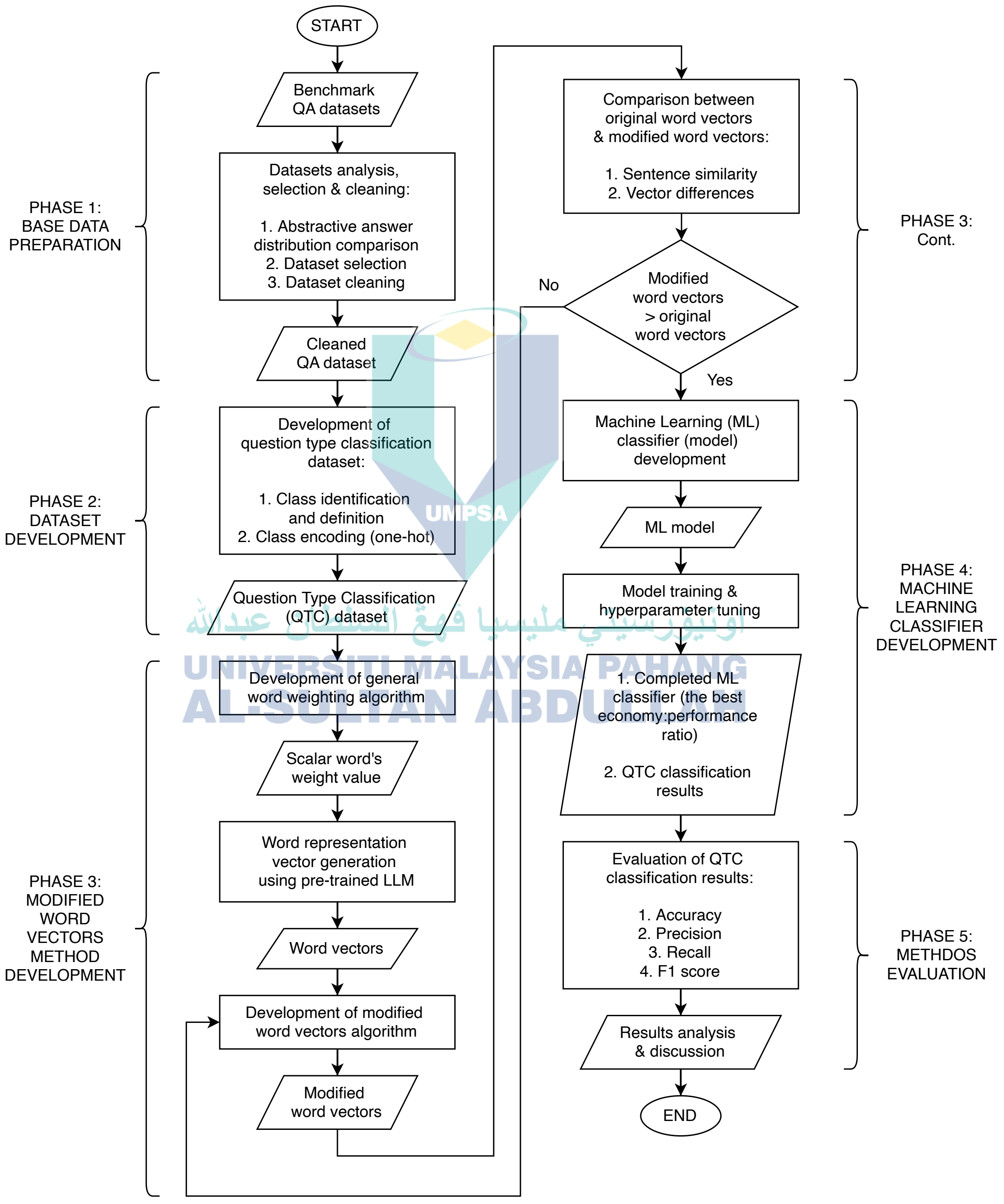

START
Benchmark QA datasets
Datasets analysis, selection & cleaning:
1. Abstractive answer distribution comparison
2. Dataset selection
3. Dataset cleaning
Cleaned QA dataset
Development of question type classification dataset:
1. Class identification and definition
2. Class encoding (one-hot)
Question Type Classification (QTC) dataset
Development of general word weighting algorithm
Scalar word's weight value
Word representation vector generation using pre-trained LLM
Word vectors
Development of modified word vectors algorithm
Modified word vectors
Comparison between original word vectors & modified word vectors:
1. Sentence similarity
2. Vector differences
Modified word vectors > original word vectors
No
Yes
Machine Learning (ML) classifier (model) development
ML model
Model training & hyperparameter tuning
1. Completed ML classifier (the best economy:performance ratio)
2. QTC classification results
Evaluation of QTC classification results:
1. Accuracy
2. Precision
3. Recall
4. F1 score
Results analysis & discussion
END
PHASE 1: BASE DATA PREPARATION
PHASE 2: DATASET DEVELOPMENT
PHASE 3: MODIFIED WORD VECTORS METHOD DEVELOPMENT
PHASE 3: Cont.
PHASE 4: MACHINE LEARNING CLASSIFIER DEVELOPMENT
PHASE 5: METHDOS EVALUATION

Figure 3.1 Research methodology flowchart

Referring to Figure 3.1, the purpose of Phase 1, the Base Data Preparation phase, is to clean and organise the initial/raw data. In Phase 2, the Dataset Development phase involves developing a new dataset using a new algorithm. Moving on to Phase 3, the Modified Word Vectors Method Development phase, the focus is on developing a new algorithm that can incorporate a scalar weight value into a vector representation of words to produce a weighted word vector of a fixed dimension size. In Phase 4, the Machine Learning (ML) Classifier Development phase, an ML classifier is created to perform the question type classification process based on the previously developed dataset and the modified word vectors. Finally, in Phase 5, the overall proposed methods within this research are assessed, with a focus on evaluating their effectiveness by comparing the produced results against the baseline score. To provide further elaboration, the following is a general summary of the planned activities for each phase.

- **Phase 1**: Base Data Preparation

As mentioned in Chapter 1, this research aims to construct a new multiclass text classification dataset that highlights distinctions in contextual representation, which will serve as the case study for this research. Furthermore, in Chapter 2, Question Classification literature is explored, and a gap within question classification datasets is identified, specifically the irrelevancy of taxonomy-based datasets and the lack of an alternative dataset that can address the question type classification problem (a multiclass classification problem that classifies question text as needing extractive or abstractive type answers). Given this gap, this research intends to develop a new question classification dataset emphasising differences in the context or type that the questions represent. Therefore, the initial phase of the research is dedicated to analysing and selecting a benchmarked Question-Answering or QA dataset as the foundation for developing the required dataset. To determine the most suitable dataset, a comprehensive comparative analysis will be conducted, with a specific focus on the distribution of abstractive answers. This analysis aims to assess the quality and diversity of answer types in each dataset. Considering the characteristics and requirements of the proposed research, one dataset will be selected as the base dataset to proceed with the subsequent phases. Before advancing to the next phase, the selected base dataset will undergo a cleaning process to remove unnecessary data and noise. Eliminating unwanted elements, the final output from this phase will be a cleaned QA dataset, ready for use in the subsequent phases of the research.

- **Phase 2**: Dataset Development

  Based on the output of the previous phase, this phase takes a cleaned QA dataset as input. From this dataset, the research aims to identify and define several classes related to abstractive answers. The class encoding process will then be performed on the dataset. Each data point will be assigned a single class that best represents it. The determination of the best class representation for each data point will be done using a specific and justifiable algorithm. The proposed algorithm will utilize one-hot encoding as the format for class encoding. This process assigns a unique binary code to each class, with a value of 1 for the corresponding class and 0 for all other classes. By using one-hot encoding, the resulting output of this phase will be a *Question Type Classification* dataset, or QTC, where each data point is associated with its corresponding class encoded in a one-hot vector format. This dataset will then be used for all further phases in this research methodology .

- **Phase 3**: Modified Word Vectors Method Development

  Before developing a new algorithm that can incorporate an external or independent scalar weight into the word representation vector, effectively modifying the word vectors' context, the scalar weight values need to be produced. Therefore, this phase will start by generating the scalar weight value that effectively signifies the general and question words within the question texts of the QTC dataset. From another perspective, this scalar weight will effectively group similar question types into the same defined class within the QTC dataset. After completing this process, word vectors will be generated for each word within the required dataset using a pre-trained LLM, and the development of the modified word vectors algorithm will commence. To recap, this research intends to develop a vanilla feed-forward Neural Network (FF-NN) model for the question-type classification process (a question-type classifier) as a means to evaluate the modified word vectors algorithm. The architecture of the vanilla FF-NN requires a fixed-length vector as its input format. Therefore, the developed modified word vectors algorithm needs to strictly adhere to this requirement, indirectly necessitating an algorithm capable of handling variable-length sentences to produce a fixed-length vector size format for the final word representation value. By the end of this phase, fixed-length weighted word vectors will have been generated for each question text in the QTC dataset, and these vectors must demonstrate improved representation of required contexts (based on a small-scale experiment) before being used to train the FF-NN model in the next phase.

- **Phase 4**: Machine Learning Classifier Development phase

  This phase focuses on building and training the question-type classifier using a machine learning algorithm, i.e., the Feed-Forward Neural Network (FF-NN) algorithm. Based on the output from the previous phases, a modified FF-NN model will be designed, taking into consideration an economical approach, to address the multiclass classification problem within the QTC dataset. Once the base model is constructed, the next step is to train the model using the QTC dataset defined classes, with the question text being the word vectors generated using the modified word vectors algorithm developed in the previous Phase 3. This question-type classifier model design and training will be done iteratively, ensuring that all hyperparameters are tested to obtain the best ratio of economy to performance results. Overall, the outcome of this phase will be a trained question-type classifier, ready for further testing and evaluation.

- **Phase 5**: Methods Evaluation

  The final phase of this research methodology focuses on validating and evaluating the overall proposed methods developed in the previous four phases. While each method has been individually validated in its respective phase, this phase aims to assess the collective contribution of each method towards achieving the end result, which is the QTC dataset classification results. There are four evaluation metrics to be analysed: Accuracy, Precision, Recall, and F1 score. To recap, this research hypothesizes that *the word representation vector derived from the LLM can be altered using an external scalar weight, which can later be used as input for an ML model to perform a text classification task*. By evaluating these four metrics, it is hoped that this hypothesis can be substantiated and provide insights into the effectiveness of the modified word vectors and the developed FF-NN model for question-type classification within the QTC dataset. This phase will involve a thorough examination of the obtained results, comparing them against baseline scores, and conducting a detailed analysis of the performance metrics. The findings will contribute to the validation of the research hypothesis and the overall success of the proposed methods for contextual text classification.

The preceding general summary for all five phases has outlined the fundamental tasks that will be undertaken for each of the defined phases in the research methodology. For a detailed implementation of this research methodology , the next subsections will explain in detail all processes carried out within each of the identified phases.

3.2.1 Phase 1: Base Data Preparation

The aim of this phase is to analyse and selects a benchmarked QA dataset to be the foundation for developing the required dataset for this research. Four benchmarked QA datasets have been selected for analysis: The Stanford Question Answering Dataset or SQuAD (Rajpurkar et al., 2016), SQuAD version 2.0 or SQuAD 2.0 (Rajpurkar et al., 2018), A Conversational Question Answering Challenge or CoQA (Reddy et al., 2019), and Question Answering in Context or QuAC (Choi et al., 2018). Overall, the creation of these datasets involved human crowd workers who were asked to produce questions based on a given context (a paragraph of text) and to produce replies by either indicating that there is no answer or by extracting an answer from the context by highlighting one continuous text span. In addition, QuAC and CoQA require workers to produce questions in the form of a dialogue where co-referencing previous interactions is possible, allowing for a conversational type of QA. QuAC and CoQA also allow direct "Yes" or "No" answers without additional explanation. Furtheremore, only in CoQA are workers allowed to edit text spans to produce abstractive answers.

The Stanford Question Answering Dataset (SQuAD) is a widely used dataset for QA systems, which contains more than 100,000 question-answer pairs, derived from Wikipedia articles. It focuses on answer extraction, where the answer to a question is a span of text within the context paragraph (Rajpurkar et al., 2016). SQuAD 2.0 is an extension of SQuAD, which includes unanswerable questions in addition to answerable ones. This dataset requires QA systems to not only extract answers from the text but also identify questions that have no answer within the given context (Rajpurkar et al., 2018). Question Answering in Context (QuAC) is a dataset that focuses on answering questions that require reasoning and context understanding. It consists of more than 14,000 information-seeking QA dialogues, where each dialogue has a context paragraph and a set of questions that follow the context (Choi et al., 2018). A Conversational Question Answering Challenge (CoQA) is a dataset that involves answering questions based on a passage of text in a conversational setting. It contains more than 127,000 questions, which are collected from more than 8,000 conversations between two crowd workers, where one worker plays the role of a "questioner" and the other plays the role of a "answerer" (Reddy et al., 2019). Overall, these datasets offer three features that make them good benchmarks for QA research: unanswerable questions, multi-turn interactions, and abstractive answers (Yatskar, 2019).

Abstractive answers are a relatively new feature in recent QA datasets, along with unanswerable questions (questions that cannot be answered due to missing or conflicting information in the context passage) and multi-turn interactions (conversational QA features). The following Figure 3.2 illustrates the distribution of QA dataset features.

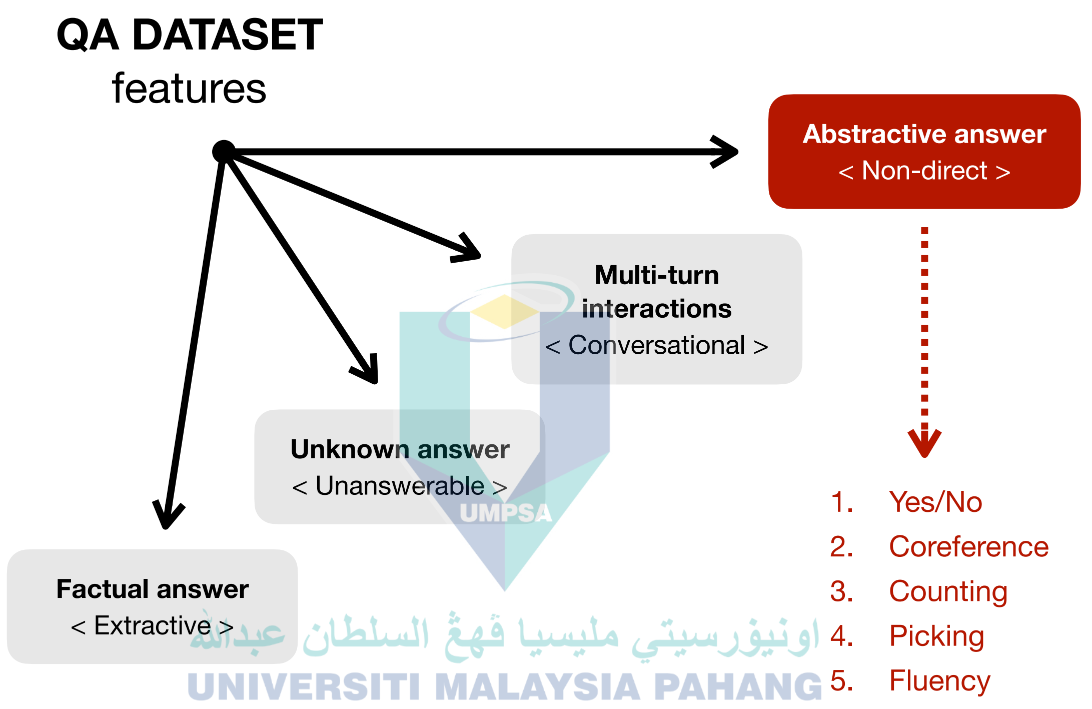


Figure 3.2 Features in QA Datasets

As depicted in Figure 3.2, there are five phenomena associated with abstractive answers: *Yes/No*, *Coreference*, *Counting*, *Picking*, and *Fluency* (S. Liu et al., 2019; Storks et al., 2019; Yatskar, 2019). For each phenomenon, *Yes/No* provides an answer to a yes or no question (e.g., Question: "Do humans need oxygen to live?", Answer: "Yes"), *Coreference* refers to an entity/object mentioned in the question or context passage (e.g., Question: "What does a lung need as an input?", Answer: "It needs oxygen"), *Counting* requires counting entities/objects in the context passage (e.g., Question: "How many main organs are in the human body?", Context: "The main organs in the human body are brain, lungs, liver, bladder, kidneys, heart, stomach, and intestines", Answer: "Eight"), *Picking* requires selecting an answer from a set defined in the question (e.g., Question: "Do humans think with their brain or heart?", Answer: "Brain"), and *Fluency* requires rephrasing the answer to be highly relevant (e.g., Question: "How do humans breathe?", Context: "The nose and mouth are two organs within the human respiratory system", Answer: "Using a respiratory system").

Based on these abstractive answer features, an analysis of the selected benchmarked QA datasets will be performed, and the best dataset with the most coverage of abstractive answers will be selected as the basis dataset to be used in the next phase of this methodology, which is the development of the new QTC dataset.

3.2.2 Phase 2: Dataset Development

The aim of this phase is to develop a Question Type Classification (QTC) dataset that classifies question text based on its paired answer text, with a focus on abstractive answer phenomena as defined in the Question-Answering (QA) NLP domain. The QTC dataset will be developed based on the benchmarked QA dataset; therefore, the analysis needed for Phase 1 needs to be completed before this Phase 2 can be started. Four benchmarked QA datasets have been selected for analysis in the previous phase: SQuAD, SQuAD 2.0, CoQA, and QuAC. Based on the conducted analysis, the CoQA dataset has been chosen as the basis for the QTC dataset to be developed in this phase. With that, the QTC dataset that will be developed in this phase will be named **QCoC** or **Question Classification of CoQA** dataset. It is to be noted that the detailed analysis result justifying the choice of CoQA to be the base for the QTC dataset will be presented in the next Chapter 4 (Results and Discussion) of this thesis.

Working with the cleaned (removed metadata dan data noise) CoQA dataset from the prior phase, the first step in this phase is to identify the classes for the QCoC dataset. Based on the distribution in the CoQA dataset, five classes have been identified for QCoC. These classes are as follows:

1. **Yes/No**: This class is intended for questions with a ground truth answer of 'yes' or 'no'. In CoQA, some yes/no answers are accompanied by additional text (e.g., "Yes, shampoo is on sale today"). Such answers will also be categorised under this class since the ground truth is still either 'yes' or 'no' (similarly for cases where the yes or no keyword appears at the end of the sentence).

2. **Unknown**: This class represents unanswerable questions based on the features of CoQA answers. Some questions in CoQA do not have a factual answer in the given context. Instead of fabricating answers, a good QA system should recognise when a question is unanswerable. Providing a made-up answer in the absence of the fact would be incorrect.

3. **Picking**: This class involves selecting one answer from multiple choices in the extracted span of the context text. Picking is similar to the Factual class (next class), but it refines the output by eliminating incorrect selections. Thus, a separate class is required for this category.

4. **Factual**: This class is intended for questions where a direct span answer can be extracted directly from the context text. It falls under the non-abstractive answer type, but since QCoC is constructed using CoQA, this class is needed to group all factual answers under one label. Additionally, the Factual class can also serve as a binary classifier, classifying answers as either extractive (factual class) or abstractive (other classes).

5. **Counting/Fluency**: This class includes answers that require counting entities/objects in the context text and answers that need to be rephrased to be highly relevant or natural. Counting and Fluency are grouped in QCoC because, syntactically, counting answers are essentially a rephrase of a text span from the context text. By merging these two phenomena, QCoC can minimize the potential for false-positive results in the classification process. Since Counting is also Fluency, falsely classifying a counting answer as a fluency answer is highly possible and should be avoided.

Based on these identified five QCoC classes, a classification algorithm is formulated. In general, this algorithm will take question text from QCoC as input, determine its class based on various conditions, and ouput the question's classified class in one-hot encoding format. The class determination conditions will also taken into accounts the original CoQA distribution of question-and-answer types which categorizes each data points into three main groups which are:

i. Whether the question is answerable or not (Answerable or Unanswerable)

ii. Whether the span (answer) is found in the context text or not (Span found or Span not found)

iii. The answer's semantic or syntactic type in one of seven categories (Named Entity, Noun Phrase, Yes, No, Number, Date/Time, Other)

To outline these distributions, Table 3.2 presents the CoQA distribution as reported by the original authors (Reddy et al., 2019). This table categorizes all items within the three main groups, separated by a row line. The first row group represents the Answerable or Unanswerable group, the second row group represents the Span found or Span not found, and the last row group represents the seven categories of semantic or syntactic types.

Table 3.1 Distribution of question and answer types in CoQA

Item	Distribution
Answerable	98.7%
Unanswerable	1.3%
Span found	66.8%
Span not found	33.2%
Named Entity	28.7%
Noun Phrase	19.6%
Yes	11.1%
No	8.7%
Number	9.8%
Date/Time	3.9%
Other	18.1%

Based on identified five classes and the distribution of question and answer types shown in Table 3.2, the following Algorithm 1 is formulated to classify question texts in CoQA into QCoC classes.

Algorithm 1 : Algorithm to classify QCoC classes

```
   Input: All question texts from the dataset
   Output: One-hot enconding for selected QCoC class
   Initialisation: Retrieve all question texts from raw QA dataset
1  foreach question text do
2      if contains 'yes'/'Yes'/'yes'/'Yes.'/'no'/'No'/'no.'/'No.' then
3          QCoC class = [1,0,0,0,0] (the Yes/No class)
4      else if contains 'unknown'/'Unknown'/'unknown.'/'Unknown.' then
5          QCoC class = [0,1,0,0,0] (the Unknown class)
6      else if contains 'or' and not contains 'what'/'where' then
7          QCoC class = [0,0,1,0,0] (the Picking class)
8      end if
9  end foreach
10 foreach question text without QCoC class do
11     get answer and span_text from original QA dataset
12     produce word vectors for both answer (A) and span_text (B)
13     calculate dot product for $A \cdot B = \|A\|\|B\| \cos\theta$
14     if $A \cdot B >$ threshold then
15         QCoC class = [0,0,0,1,0] (the Factual class)
16     else
17         QCoC class = [0,0,0,0,1] (the Counting/Fluency class)
18     end if
19 end foreach
20 return QCoC class (one-hot encoding) for each question text
```

Algorithm 1 is designed to classify each question text in CoQA into one of the five QCoC classes. Examining the algorithm steps, steps 1 to 9 are a straighforward process, as written, to classify question text into **Yes/No**, **Unknown** and **Picking** classes. For the other two classes (**Factual** and **Counting/Fluency**) in steps 10 through 19, a more in-depth semantical analysis is conducted. Specifically, the *answer* and *span_text* corpus from CoQA are retrieved, and word vector embeddings are produced for both corpuses using the Universal Sentence Encoder (USE) (Cer at al., 2018) LLM. To elaborate, *span_text* in CoQA is a piece of text from a long story/context paragraph, representing the information used to generate the paired *answer* text. After the embeddings are generated, the dot product for both the answer and span_text vector values is calculated using the following Equation 3.1:

$$A \cdot B = \|A\|\|B\| \cos\theta \tag{3.1}$$

In Equation 3.1, A and B are word embeddings of the answer and span_text in a 100-dimensional vector format. The dot product (using cosine similarity) of A and B will provide an indicator of how similar both sentences are in terms of their semantic representation (higher dot product signifies higher similarity). Based on the calculated dot product values, the balanced question text that hasn't been assigned a QCoC class will be distributed between the **Factual** and **Counting/Fluency** classes using a threshold value defined based on the overall distribution of the dot product and the original CoQA's question text distribution, showcased in the following Table 3.2 and the previous Table 3.1, respectively.

Table 3.2 The USE dot product distribution result

Item	**Result**
Maximum	125.44
Minimum	89.52
Overall percentage distribution	>= 123.64 (31.335%)
**Dividing the maximum and minumum values into 20 segments*	121.85 − 123.63 (12.547%)
	120.05 − 121.84 (11.497%)
	118.26 − 120.04 (8.438%)
	116.46 − 118.25 (5.740%)
	114.66 − 116.45 (3.713%)
	112.87 − 114.65 (2.366%)
	111.07 − 112.86 (1.374%)
	109.28 − 111.06 (0.839%)
	107.48 − 109.27 (0.458%)
	105.68 − 107.47 (0.247%)
	103.89 − 105.67 (0.132%)
	102.09 − 103.88 (0.080%)
	100.30 − 102.08 (0.031%)
	98.50 − 100.29 (0.023%)

	96.71 − 98.49 (0.009%)
	94.91 − 96.70 (0.009%)
	93.11 − 94.90 (0.002%)
	91.32 − 93.10 (0.000%)
	< 91.31 (0.002%)

As mentioned, the classification of the **Yes/No**, **Unknown**, and **Picking** classes is relatively straightforward using steps 1 to 9 in the presented Algorithm 1. However, for the **Factual** and **Counting/Fluency** classes, the classification is determined based on the USE dot product distribution. This distribution is obtained by analysing the minimum and maximum dot product values (89.52 minimum and 125.44 maximum, as presented in Table 3.2), which then allows the data to be divided into 20 range segments. Based on these 20 segments and the CoQA question-answer distribution presented in Table 3.1, 64,589 (55.38%) data points are classified as **Factual** (the sum of the three highest segments), while 27,364 (23.47%) data points are classified as **Counting/Fluency** (the sum of the 17 lowest segments).

As previously mentioned, a higher value of the USE dot product signifies a higher semantic similarity between the *answer* and the *span_text*. Factual answers in a QA dataset (i.e., the CoQA dataset) are highly extractive, meaning that the answer to a Factual question is likely to be similar in terms of both semantic and syntactic aspects to the span text retrieved from the context text. This scenario results in a higher USE dot product value when calculated between the *answer* and the *span_answer* text. Regarding the justification for "the sum of the three highest segments and the sum of the 17 lowest segments in Table 3.2," this action is based on the distribution of CoQA answer types presented in Table 3.1. Detailed analysis regarding this action will be further presented in the next chapter of this thesis (Chapter 4: Results and Discussion).

Using the proposed Algorithm 1, a new dataset named QCoC (Question Classification of CoQA) will be developed. This dataset will later be used to evaluate the modified word vectors method (which will be developed in the next phase of this research methodology) through a machine learning classifier that will be developed in further Phase 4 of this research methodology. Because the classifier will be developed using the Feed-Forward Neural Network (FF-NN) algorithm, its inputs need to be in numerical format, which for the QCoC dataset is the word embeddings/vectors for each question text. Directly using word vectors for the training of the classifier seems to produce inconclusive results due to high similarities between each vector. To visualize this problem using a smaller case study, five semantically close questions are defined, and word vectors using USE are generated for each question. Following Figure 3.3 visualizes those generated vectors (100-dimensional) in a 3D line graph format.

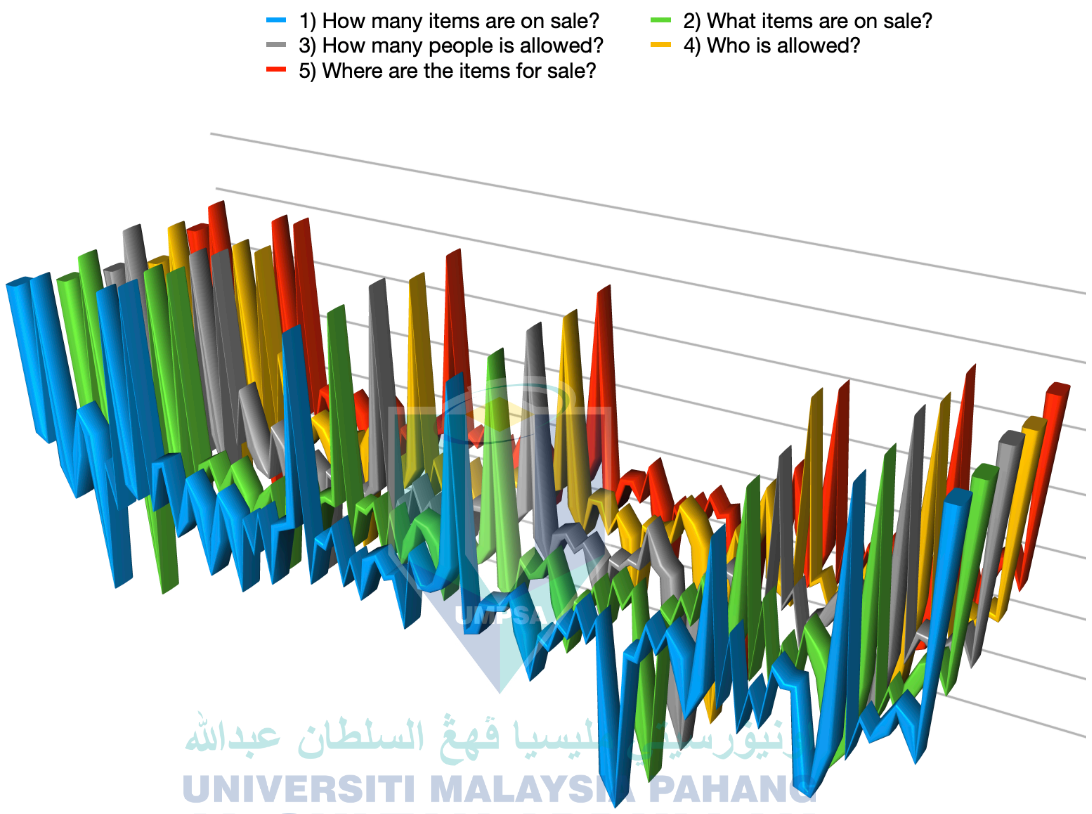


Figure 3.3 Line graph for five questions of USE-generated vectors

Referring to Figure 3.3, the generated word vectors from USE LLM are shown to be closely similar, justifying the need to fine-tune the LLM parameters to modify the contextual weight of each vector for the text classification process. With such being the case, the next Phase 3 of this research will propose a modified word vectors method that is able to adjust the weight of the LLM-generated vectors using an external scalar value. This method is required in order to prepare the word vectors as input to the FF-NN classifier that will be developed in the next Phase 4 of this research methodology.

3.2.3 Phase 3: Modified Word Vectors Method Development

The aim of this phase is to develop a method that can modify the LLM-generated word vectors into vector values that signify the desired weight target. In this research, the desired weight target is words that are able to differentiate question texts, enabling proper classification into one of the QCoC classes. Overall, Phase 3 will involve two main activities: 1) Developing an algorithm to produce scalar weight that signifies the occurrences of question/general words, and 2) Developing an algorithm to take the scalar weight value and modify the LLM-generated word vectors to produce vectors that are weighted towards the scalar weight.

Regarding the first activity, the generated scalar weight value should emphasise the question words, which are key weighting factors in differentiating question types. To generalize this process, activity 1 should also consider other words used in the question texts. As mentioned in the literature of this research, identifying question/general word occurrences in question texts is useful for determining whether a question requires a direct factual answer or an indirect abstractive answer. Therefore, the first activity will focus on developing an algorithm that can represent the usage of general words in the question text within the benchmarked QA datasets.

For the second activity, the focus will be on developing an algorithm to modify LLM-generated word vectors into weighted fixed-length vector values. Weighted in this case is towards the embedded scalar value, and fixed-length in this case refers to the same length vector values for sentences with varying word counts. As mentioned, the architecture of the vanilla FF-NN requires a fixed-length vector as its input format, indirectly necessitating an algorithm capable of handling variable-length sentences to produce a fixed-length vector size format for the final word representation value.

- **Activity 1**: Development of general word weighting algorithm

This activity aims to develop an algorithm that can generate a scalar weight value capable of effectively emphasising the usage of general or question words in a sentence. This algorithm will utilize the Term Frequency (TF) formulation as the base formula, which calculates the frequency or usage of general/question words in the given dataset. To further validate this algorithm's result, the produced scalar weight value must also be scalable to any document or library size. With this generalization, other cases can also implement this method by synchronising it with the selected domain's corpus, such as using medical-specific words instead of question words for the medical domain dataset. The algorithm designed for this purpose is outlined in the following Algorithm 2.

ALGORITHM 2 : ALGORITHM TO GENERATE WORD'S SCALAR WEIGHT

```
    Input: All question texts from the dataset
    Output: token-weight pair
    Initialisation: Retrieve all question texts from raw QA dataset
1   Combine all question text into one long word sequence
2   Tokenise the long sequnece using WordPiece tokeniser
3   foreach token in the long sequence do
4       if token = '?' or '"' or ',' or '.' then
5           remove token
6       else if token ≠ alphabets (i.e. number only) then
7           remove token
8       else if token = previous tokens (duplicate) then
9           remove token
10      else
11          token += Unique Token (UT)
12      end if
13  end foreach
14  foreach UT do
15      Term Frequency (TF) = UT counts in long sequence
16      weight (i.e. TF percentage) = $(TF / UT) \times 100$
17  end foreach
18  return list of token-weight pair
```

As mentioned, this algorithm will be validated by its capability to produced weight values that are scalable to any document or library size. To demonstrate this capability, two separate but interconnected experiments will be conducted. The first experiment, denoted as "experiment A," will solely use the CoQA dataset, while the second experiment, "experiment B," will involve the combination of CoQA, SQUAD, and QuAC datasets. Both experiments will exclusively use the *Question texts* from each dataset, comprising a combination of the *Question texts* from the training and validation datasets. The primary distinction between these two experiments lies in the scale of the datasets. Experiment A uses only the CoQA dataset, which has a total of 662,607 tokens (17,404 unique tokens), while experiment B involves a combination of the CoQA, SQUAD, and QuAC datasets, resulting in a total of 2,833,785 tokens (24,837 unique tokens). The difference between the total tokens in experiment A and experiment B is 2,171,178, while the difference in unique tokens is 7,433. These differences in dataset sizes will enable the assessment of the proposed algorithm's scalability across various document or library sizes.

Referring to the Algorithm 2, the WordPiece tokeniser (Devlin et al., 2018) is used to segregate the tokens from the original input text (step 2 of the algorithm). Tokenising the question text is a necessary process as the weight will be associated with each token, where one sentence (question texts) may have multiple tokens. After the tokenization process is completed, several irrelevant tokens, such as symbols and numbers, will be deleted before proceeding with the TF calculation. Upon executing the whole algorithm, the Term Frequency (TF) percentage value is calculated for each Unique Token (UT), referring to tokens that are not symbols, numbers, or duplications of other tokens. This TF percentage value is denoted as the 'weight' value (step 16 in the algorithm) and is incorporated into the token-weight pair data, which constitutes the algorithm's output.

This *Activity 1: Development of general word weighting algorithm*, aims to develop an algorithm capable of calculating and producing scalar weight values to emphasise general/question words in a given text. The motivation behind this algorithm is to address the issue where the weight of general/question words is overshadowed by the weight of context words in the contextualised word vectors produced by the pre-trained LLM. To generate the necessary scalar weight value, this algorithm employs the UT distribution percentage, which is expected to emphasise the weightage of general/question words without overshadowing the overall contextual semantics of the given text. To thoroughly evaluate this expectation, these scalar weight values will be incorporated into the LLM-generated word vectors, a task which will be performed in the next activity (Activity 2) within Phase 3 of this research methodology.

- **Activity 2**: Development of modified word vectors algorithm

This activity aims to develop an algorithm that can incorporate an external scalar weight into the word representation vector generated from the LLM. The scalar vector from previous Activity 1 will be used together with LLM-generated word vectors from USE LLM as inputs to this algorithm. The output of this algorithm is a set of weighted fixed-length word representation vectors for question texts, which will later serve as input to the FF-NN question type classifier in Phase 4 of this research methodology. To produce the fixed-length vector, original 100-dimensional USE LLM word vectors for each word in a sentence (question text) will be utilised to represent the entire sentence. Maintaining this fixed-length vector will be challenging due to the varying length of sentences (different word counts). Therefore, four algorithms are proposed for the experiment to achieve the aim of this Phase 3. The four algorithms are presented in mathematical form as follows:

1. $$wv = \left[\left[\left(v^i \cdot gw^i\right)\left(v^{i+1} \cdot gw^{i+1}\right)^T\right]\left(v^{i+2} \cdot gw^{i+2}\right)^T\right] \ldots \left(v^n \cdot gw^n\right)^T$$

2. $$wv = vs \cdot \prod_{i=1}^{n} gw^i$$

3. $wv = vs \cdot [gw^1, gw^2, gw^3 \ldots\ gw^n]$, where $gw^i = 1$ if $gw^i =$ no value

4. $$wv = \frac{\sum_{i=1}^{n}(v^i \cdot gw^i)}{n}$$

where:

- $wv =$ weighted fixed-length vector
- $v =$ vector embedding for each token
- $i =$ token's index
- $gw =$ general weight or g-weight value for each token (in scalar format)
- $n =$ total token counts in the question text
- $vs =$ vector embedding for the input sentence

Given equation one to four above represent different proposal to calculate the weighted fixed-length vector, denoted as wv, where v is the vector embedding/representation for each token produced by the USE LLM, at the token's index i. The variable gw denotes the general weight or g-weight value associated with each token, which is the result of Algorihtm 2 in Activity 1 of this Phase 3. In cases where a token/word lacks a corresponding g-weight value, the algorithm will assign $gw =$

0.00001, which is lower than the lowest g-weight value observed in previous experiment (0.00015 for Experiment A and 0.00018 for Experiment B). The variable n represents the total token count in the question text, which serves as the input for this algorithm. Lastly the variable vs denotes the vector embedding for the entire input sentence or question text, where the USE LLM is used to produce the whole sentence embedding as opposed to embedding for each token. To further interpret these four algorithms, Algorithm 3 to 6 will outline these algorihtms in pseudocode format.

ALGORITHM 3 : ALGORITHM TO GENERATE WEIGHTED FIXED-LENGTH VECTOR – V1

```
       Input: All question texts from the dataset
       Output: Weighted fixed-length vector for each question text
       Initialisation: Retrieve all question texts from classification dataset
1      foreach question text do
2          | tokenise using WordPiece tokeniser
3          | foreach token do
4          |     | generate $v$ (vector embedding) using USE LLM
5          |     | get $gw$ (g-weight) scalar value from g-weight lookup table
6          |     | calculate $wt^i$ (weighted token for each token i) as $v \cdot gw$
7          |     | if the first token or $i = 1$ then
8          |     |     $wv$ is equal to $wt^i$
9          |     | else
10         |     |     calculate $wv$ as $wv$ multiply by $wt^i$
11         | end foreach
12     end foreach
13     return $wv$ for each question text
```

Algorihtm 3 showcases Variation 1 (V1) of the proposed algorihtm to generate the weighted fixed-length vector. In summary, this algorithm will take all question texts from the given dataset, tokenise them using WordPiece tokeniser (Devlin et al., 2018), and loop through each segmented token to perform several processes sequentially: generate the vector embedding (v) of that token using USE LLM in 100-dimensional format, obtain the g-weight (gw) value for that token from the lookup table, calculate the weighted token (wt^i) as $v \cdot gw$, and finally calculate the weighted fixed-length vector

(wv) as $wv \cdot wt^i$. These processes will be performed iteratively until all tokens within the question text are processed, then the algorithm moves on to the next question text in the given dataset. Ultimately, wv for all question texts will be generated.

ALGORITHM 4 : ALGORITHM TO GENERATE WEIGHTED FIXED-LENGTH VECTOR – V2

```
     Input: All question texts from the dataset
     Output: Weighted fixed-length vector for each question text
     Initialisation: Retrieve all question texts from classification dataset
1    foreach question text do
2        g-weightSum = 1
3        tokenise question text using WordPiece tokeniser
4        foreach token do
5            get $gw$ (g-weight) scalar value from g-weight lookup table
6            g-weightSum = g-weightSum · $gw$
7        end foreach
8        generate $vs$ (vector sentence embedding) using USE LLM
9        calculate $wv$ as $vs$ · g-weightSum
10   end foreach
11   return $wv$ for each question text
```

Algorihtm 4 showcases the Variation 2 (V2) of the proposed algorihtm to generate the weighted fixed-length vector. Initially similar to V1, this algorithm will take all question texts from the given dataset, tokenise them using WordPiece tokeniser (Devlin et al., 2018), and loop through each segmented token to perform several processes sequentially. Within this loop, the g-weight (gw) value for each token will be obtained from the lookup table, and the sum of gw will be calculated as *g-weightSum* = *g-weightSum* · gw. After all tokens have been processed, a vector sentence embedding (vs) is generated using USE LLM in 100-dimensional format, and the weighted fixed-length vector (wv) is calculated as vs · *g-weightSum.*These processes will then be repeated for the next question text untill all question texts within the given dataset are processed and wv for all question texts have been generated.

ALGORITHM 5 : ALGORITHM TO GENERATE WEIGHTED FIXED-LENGTH VECTOR – V3

```
    Input: All question texts from the dataset
    Output: Weighted fixed-length vector for each question text
    Initialisation: Retrieve all question texts from classification dataset
1   foreach question text do
2       set g-weightEmbedding[n] = [1,1,1, … ,1]
3       tokenise question text using WordPiece tokeniser
4       i = 0
4       foreach token do
5           get gw (g-weight) scalar value from g-weight lookup table
6           set g-weightEmbedding[i] = gw
            i++
7       end foreach
8       generate vs (vector sentence embedding) using USE LLM
9       calculate wv as vs · g-weightEmbedding[n]
10  end foreach
11  return wv for each question text
```

Algorihtm 5 showcases the Variation 3 (V3) of the proposed algorihtm to generate the weighted fixed-length vector. Also initially similar to V1 and V2, this algorithm will take all question texts from the given dataset, tokenise them using WordPiece tokeniser (Devlin et al., 2018), and loop through each segmented token to perform several processes sequentially. But before entering the tokens loop, a *g-weightEmbedding[n]* for each n question text will be initialised as a vector of 100-dimensional space with an integer value of '1' for each space. Then for each token, the g-weight (gw) value will be obtained from the lookup table, and directly the gw scalar value will be placed on the *g-weightEmbedding[n][i],* with i being the index of the token in the question text. This process will be performed iteratively until all tokens within the question text are processed. Then, a vector sentence embedding (vs) is generated using USE LLM in 100-dimensional format, and the weighted fixed-length vector (wv) is calculated as $vs \cdot$ *g-weightEmbedding[n]*.These processes will then be repeated for the next question text untill all question texts within the given dataset are processed and wv for all question texts have been generated.

ALGORITHM 6 : ALGORITHM TO GENERATE WEIGHTED FIXED-LENGTH VECTOR – V4

```
     Input: All question texts from the dataset
     Output: Weighted fixed-length vector for each question text
     Initialisation: Retrieve all question texts from classification dataset
1    foreach question text do
2        set total_wt[n] = [0,0,0, … ,0]
3        tokenise using WordPiece tokeniser
4        foreach token do
5            generate $v$ (vector embedding) using USE LLM
6            get $gw$ (g-weight) scalar value from g-weight lookup table
7            calculate $wt^i$ (weighted token for each token i) as $v \cdot gw$
8            calculate total_wt[n] = total_wt[n-1] + $wt^i$
9        end foreach
10       calculate $wv$ as total_wt[n] divided by token counts (average of $wt$)
11   end foreach
12   return $wv$ for each question text
```

The last Variation 4 (V4) of the proposed algorihtm to generate the weighted fixed-length vector is showcases as Algorithm 6. Also initially similar to previous variations, this algorithm will take all question texts from the given dataset, tokenise them using WordPiece tokeniser (Devlin et al., 2018), and loop through each segmented token to perform several processes sequentially. Before entering the tokens loop, a *total_wt[n]* for each n question text will be initialised as a vector of 100-dimensional space with an integer value of ‘0’ for each space. Then for each token, the vector embedding (v) of that token will be generated using USE LLM in 100-dimensional format, and the g-weight (gw) value for that token will be obtained from the lookup table. Next, the weighted token for each token i (wt^i) is calculated as $v \cdot gw$, and subsequently the *total_wt* is calculated as *total_wt[n]* = *total_wt[n-1]* + wt^i. After all tokens have been processed, the weighted fixed-length vector () will be calculated as *total_wt[n]* divided by the token count for the current question wv text, which is basically an average of all wt^i. These processes will then be repeated for the next question text untill all question texts within the given dataset are processed and wv for all question texts have been generated.

To validate the results of all proposed algorithms, the weighted fixed-length vectors (wv) for five question texts are compared. The purpose of this process is to verify the accuracy with which each proposed algorithm weights the word vectors using the given external scalar weight value. Table 3.3 outlines the five question texts that will serve as test cases for this experiment. For all five cases, two types of word vectors are calculated: those with the g-weight implementation (type A with four variations: A-V1, A-V2, A-V3, and A-V4) and those without the g-weight implementation (type B). For type A (A-V1 to A-V4), the produced word vectors are the wv from the previously presented four algorithms, and for type B, the produced word vectors are the original outputs from USE LLM.

Table 3.3 Five test cases for the word weighting method experiment

No	Question text	General words	Context words
1	how many items are on sale?	how, many, are, on	items, sale
2	what items are on sale?	what, are, on	items, sale
3	how many people is allowed?	how, many, is	people, allowed
4	who is allowed?	who, is	allowed
5	where are the items for sale?	where, are, the, for	items, sale

To validate whether the proposed algorithm has successfully transformed the original USE LLM word vectors into weighted fixed-length vectors, two comparisons will be conducted for types A and B. The first comparison involves measuring sentence similarity through dot product calculations, while the second comparison involves measuring vector differences (in real numbers) through direct matrix subtraction. Based on these comparisons, one of the variations of algorithm A (A-V1, A-V2, A-V3, or A-V4) that is capable of highlighting the weightage of the general words will be selected as the algorithm that will be used in the overall modified word vectors method. As a whole process, this modified word vectors method will use the pre-trained LLM to generate word vectors, obtain the general weight or g-weight scalar value from the precalculated lookup table (precalculated using Algorithm 2 in this phase's Activity 1), and finally modify the original LLM-generated word vectors using one of the modified word vectors algorithms (Algorithm 3 or A-V1, Algorithm 4 or A-V2, Algorithm 5 or A-V3, or Algorithm 6 or A-V4) proposed in this phase's Activity 2. The result from this method, which is the weighted fixed-length vector, will then be used as input to the FF-NN that will be developed in the next Phase 4 of this research methodology.

3.2.4 Phase 4: Machine Learning Classifier Development phase

The aim of this phase is to build and train a Machine Learning (ML) classifier for a question-type classification task. As mentioned before, the ML classifier will be built using a vanilla/base feed-forward Neural Network (FF-NN) algorithm, aiming for the most economical or least computing resource model that suits multiclass classification problems. Given this aim, this phase will involve sequentially building the classifier, starting from the most basic setup or minimum parameter count towards the most advanced setup or maximum parameter count, attempting to achieve the best ratio of economy to performance results.

Starting with the most basic FF-NN setup (one input, one hidden, and one output layer), this experiment will test various setups to find the best balance between economy and performance. To achieve this, various variables, including the FF-NN setup and hyperparameters, will be experimented with. The following is a list of those setups and hyperparameters to be explored:

- Hidden layer size
- Activation function
- Gradient descent type
- Loss function
- Optimizer function
- Learning rate
- Batch size
- Epoch size
- Dropout usage

The primary objective of this experiment is to find the most efficient and economical configuration for the question-type classifier ML model, while still ensuring satisfactory classification accuracy for the given contextual text data. This will be achieved by systematically varying the parameters and analysing the resulting performance through a set of sequential experiments. To ensure fair and consistent comparisons, all experiments will be conducted using the same hardware and software setup. Table 3.9 provides an overview of the specific details of this setup.

Table 3.4 Hardware and software setup for ML classifer experiment

Hardware	Software
- CPU: 2.3GHz Quad-Core Intel Core i7 - Memory: 32GB 3733MHz - GPU: Intel Iris Plus Graphics 1536MB	- Tensorflow Javascript (TFJS) - WebGL runtime through Google Chrome - macOS version 11.2.3

Although multiple experimental setups will be conducted for this experiment, the base feed-forward Neural Network (NN) architecture for the multiclass classification problem will remain consistent. Figure 3.5 illustrates this base model architecture.

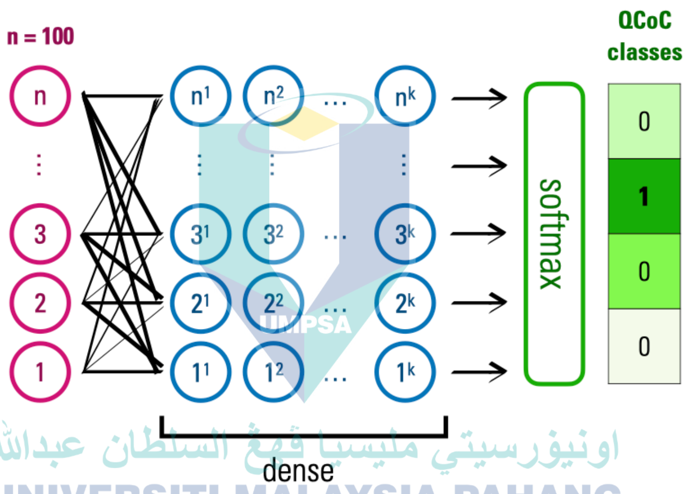


Figure 3.4 Base feed-forward NN model design for ML classifier experiment

Referring to Figure 3.5, the model's main constant design consists of the input and output layer sizes, which comprise 100 nodes and four nodes, respectively. As mentioned earlier, the input size is determined by the vector embedding produced by USE LLM (Cer et al., 2018), while the output size corresponds to the four QCoC classes, as this is a supervised ML model. Notably, the original QCoC class includes five categories; however, this experiment only employs four classes, excluding the "Unknown" class. The rationale behind this decision will be provided in the next paragraph. Additionally, the usage of the softmax activation function for the output layer remains consistent throughout the experiments. This choice is well-suited for performing the multiclass classification process involving the four QCoC classes. The softmax function is denoted by following Equation 3.2.

$$softmax(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad 3.2$$

where:

- $\vec{z}$ = input vector from previous NN nodes
- e^{z_i} = exponential function for an input vector
- K = number of classes (four QCoC classes for this experiment)
- e^{z_j} = exponential function for output vector

For all experimental setups, 98.76% of the QCoC dataset will be used. As a recap, the QCoC dataset comprises five classes, namely *Yes/No* (19.27% of the total data points), *Unknown* (1.24%), *Picking* (0.64%), *Factual* (55.38%), and *Counting/Fluency* (23.47%). For this experiment, the *Unknown* class (1.24% or 1450 out of the total 116630 QCoC data points) will not be utilised, leaving only four classes with 98.76% or 115180 QCoC data points. The reason for disregarding the *Unknown* class is that the classifier's objective is to classify question text based on general and question words. Since the Unknown class's answer does not correlate with its question type (any question type can be classified as *Unknown* if the required answer is not presented in the given context), including this class in the classifier training dataset would introduce an outlier that could hinder the classifier from learning the pattern of other classes effectively.

To reiterate, the problem being addressed in this research is the multiclass classification of the QCoC dataset. To assess the performance of the classifier, the F1 score is chosen as the primary metric. The F1 score is a well-established and widely used metric for multiclass classification tasks, especially in scenarios with imbalanced class distributions such as the QCoC dataset. With the general target of achieving the highest F1 score in the shortest possible training duration, the following sequential experiments are conducted. These experiments are sorted in ascending order, starting from the first experiment and progressing towards the last.

i. Determining hidden layer node counts for one hidden layer

ii. Determining the best combination of hidden layer and epoch size

iii. Determining the best activation function

iv. Determining the best optimizer function

v. Determining the best combination of batch and epoch size

vi. Determining the best loss function

vii. Determining the best combination of the loss function and epoch size

viii. Determining the best learning rate

In all experiments, a 0.2 validation split is performed for the training dataset, which means 80% of the QCoC data points are used for training, and the remaining 20% are used for validation. Once the training and validation process is completed, the trained model will be evaluated using precision, recall, and F1 metrics on the entire QCoC dataset, which comprises a total of 115,180 data points.

The baseline F1 score for the QCoC dataset is 0.561, which is calculated based on the highest F1 score obtained from the "Factual" class, considering true positive, true negative, and false negative values. With this defined baseline F1 score, sequential experiments will be conducted, where each setup will be evaluated based on its F1 score. The setup that achieves the highest F1 score will be selected as the base setup for subsequent experiments. To further elaborate on the evaluation methods for this research, Phase 5 will explain in detail all evaluation metrics that will be used in evaluating the trained ML classifier developed from this phase.

3.2.5 Phase 5: Methods Evaluation

The aim of this final phase is to validate and evaluate all proposed and developed methods from the previous four phases. While each method has been individually validated in its respective phase, this phase aims to assess the collective contribution of each method towards achieving the end result, which is the classification of the QCoC dataset using the developed ML classifier from Phase 4. Specifically, the evaluation will focus on the classifier's ability to predict classes for the QCoC dataset, aiming to justify the research hypothesis, which posits that *the word representation vector derived from the LLM can be altered using an external scalar weight, which can later be used as input for an ML model to perform a text classification task.*

Overall, there are four evaluation metrics to be analysed: Accuracy, Precision, Recall, and F1 score. The mathematical forms of these metrics are as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad 3.3$$

$$precision = \frac{TP}{TP + FP} \quad 3.4$$

$$recall = \frac{TP}{TP + FN} \quad 3.5$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \quad 3.6$$

where:

- TP = True Positive value (correct prediction of positive/truth class)
- TN = True Negative value (correct prediction of negative/false class). In this experiment, TN is always 0 because there is no binary negative class. After all, multiclass classification makes other not-positive classes as FP (False Positive).
- FP = False Positive value (wrong prediction of other classes). The system predicted other class while it should be the truth class.
- FN = False Negative value (wrong prediction of truth class). The system predicted truth class while it should be in another class.

In general, the accuracy metric measures the overall performance of the model (correct predictions divided by all predictions). The precision metric evaluates the model's ability to predict the true class (correct predictions of the true class divided by all predictions of that class). The recall metric assesses the model's capability to identify the wrong prediction of the true class (correct predictions of the true class divided by all actual instances of that class). The F1 metric produces a harmonic mean of precision and recall, providing a balanced measure of both metrics.

To recap, the QCoC dataset is imbalanced, with the following class distribution: *Yes/No* class: 22,478 data points (19.27% of the total), *Unknown* class: 1,450 data points (1.24%), *Picking* class: 749 data points (0.64%), *Factual* class: 64,589 data points (55.38%), and *Counting/Fluency* class: 27,364 data points (23.47%). As mentioned in the previous Phase 4 of this research methodology , the *Unknown* class is disregarded in this experiment due to the ambiguous structure of its questions (any question type can be classified as *Unknown* if the required answer is not presented in the given context). Consequently, the total number of data points to be tested for this experiment is 115,180, which accounts for 98.76% of the total 116,630 QCoC data points. Based on the distribution of the four remaining classes (*Yes/No*, *Picking*, *Factual*, and

Counting/Fluency), the baseline accuracy and F1 score can be calculated using the highest element in one class, which is the *Factual* class with 64,589 or 56.08% of the total 115,180 data points. Therefore, the baseline accuracy and F1 score for this model are 0.390 and 0.561, respectively (assuming the model predicted all data points as the *Factual* class).

After the completion of this final phase, the research methodology will be concluded, and the result for tasks and experiements within all phases will be presented and discussed. All expected results and discussion will be summarised in the next section of this chapter.

3.3 Summary

This chapter presented a five-phase research methodology structured for this study, comprising 1) Base Data Preparation, 2) Dataset Development, 3) Modified Word Vectors Method Development, 4) Machine Learning Classifier Development, and 5) Methods Evaluation. Phase 1 focuses on cleaning and organising raw data, which begins with selecting a benchmarked Question-Answering (QA) dataset. In Phase 2, a cleaned QA dataset from phase 1 is used to identify and define classes related to abstractive answers, developing a Question Type Classification (QTC) dataset named QCoC or Question Classification of CoQA. Phase 3 involves generating scalar weight values to group similar question types, then developing an algorithm to modify word vectors for each question in the QCoC dataset. Phase 4 builds and trains a Feed-Forward Neural Network (FF-NN) model for question-type classification using the QCoC dataset and modified word vectors as inputs to the training process. Finally, Phase 5 evaluates the overall methods by comparing classification results against baseline scores using evaluation metrics Accuracy, Precision, Recall, and F1 score. With this presented research methodology, this study aims to justify and substantiate the hypothesis that altering word representation vectors using external scalar weights can enhance the contextual text classification tasks.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter aims to provide a comprehensive analysis and interpretation of the results obtained from tasks and experiments conducted throughout the five-phase research methodology. To summarise, the expected results for each phase of the research methodology are as follows:

- **Phase 1: Base Data Preparation**. A cleaned and organised dataset derived from one of multiple benchmarked QA dataset. This dataset will later be used as the basis for dataset development in the next phase of the methodology.
- **Phase 2: Dataset Development**. A Question Type Classification (QTC) dataset, which is a multiclass classification dataset with classes related to abstractive answers phenomena in QA datasets.
- **Phase 3: Modified Word Vectors Method Development**. Two results are expected from this phase, which comprises two main activities. The first result is a general word weighting algorithm, and the second result is a modified word vectors algorithm. Combining both algorithms, experiments conducted in this phase will produce a uniform method for modifying LLM-generated word vectors.
- **Phase 4: Machine Learning Classifier Development**. A machine learning classifier developed using the Feed-Forward Neural Network (FF-NN) algorithm for the multiclass classification of the QCoC dataset. This classifier should be iteratively trained with different setups and hypeparameter tuning to achieve the best ratio of economy to performance results.

- **Phase 5: Methods Evaluation**. An evaluation of previously developed ML classifier using Accuracy, Precision, Recall, and F1 score. The findings will contribute to the validation of the research hypothesis and the overall success of the proposed methods for contextual text classification.

Based on these expected results, the following section will present a result analysis for each of the methodology phases, followed by a discussion of findings in the subsequent section that will relate the overall outcomes of those results to the defined objectives of this study.

4.2 Results Analysis

As a general recap of the five-phase methodology used in this study: the first phase involves identifying the best dataset to be used as the basis for dataset development in the second phase. The third phase entails developing algorithms to produce scalar weight values that best represent the selected case study, followed by developing algorithms to modify the raw LLM-generated word vectors to embed the scalar weight values. The fourth phase focuses on developing an ML model for text classification of the developed dataset, and finally, the fifth phase evaluates the performance of the ML model to reflect the effectiveness of the developed algorithms in the third phase of the research methodology.

4.2.1 Phase 1: Base Data Preparation

The aim of this phase is to analyse and select a benchmarked QA dataset to serve as the foundation for developing the required dataset for this research. Initially, four benchmarked QA datasets have been selected for analysis: The Stanford Question Answering Dataset or SQuAD (Rajpurkar et al., 2016), SQuAD version 2.0 or SQuAD 2.0 (Rajpurkar et al., 2018), A Conversational Question Answering Challenge or CoQA (Reddy et al., 2019), and Question Answering in Context or QuAC (Choi et al., 2018). Later in the deeper analysis, SQuAD is neglected as its contents are also within the SQuAD 2.0 dataset. An overview analysis demonstrated that these datasets are primarily structured around the *Factual answer* feature, where the answer can be directly extracted from the given context. For the *Unknown answer* feature, only SQuAD 2.0 and QuAC contain a significant amount of it, with SQuAD 2.0 simulating questioner confusion and

QuAC focusing on missing information (Yatskar, 2019). In regards to the *Multi-turn interactions feature*, only QuAC and CoQA are presented with this feature, as both datasets are structured around conversational QA interactions. Regarding the *Abstractive answer* feature, only QuAC and CoQA include it, with most of it being the Yes/No phenomenon. However, QuAC does not include other phenomena, while CoQA also includes Coreference and Fluency (Yatskar, 2019). The coverage of abstractive answer phenomena in these datasets is outlined in Table 4.1.

Table 4.1 Abstractive answer implementation in SQuAD 2.0, CoQA and QuAC

Abstractive answer phenomena	**SQuAD 2.0**	**CoQA**	**QuAC**
Yes/No	No	Yes	Yes
Coreference	No	Yes	No
Counting	No	Yes	No
Picking	No	Yes	No
Fluency	No	Yes	No

Source: Yatskar (2019).

Based on this analysis, the CoQA dataset has been chosen as the basis for developing the QTC dataset for this research, primarily due to its wide coverage of abstractive-type answers. Henceforth, the QTC dataset will be referred to as the **QCoC (Question Classification of CoQA)** dataset to reflect CoQA as the foundational dataset for the QTC dataset. Before delving into the QCoC dataset development process (Phase 2 of this research), the CoQA dataset will be cleaned, with metadata and data noise removed. The result of this Phase 1 (Base Data Preparation) will be the cleaned CoQA dataset, which will then serve as the input for the subsequent Phase 2 of this research methodology .

4.2.2 Phase 2: Dataset Development

Using the cleaned CoQA dataset from the previous phase as a foundation, the QCoC dataset is being developed in this phase. Five QCoC classes have been defined: Yes/No, Unknown, Picking, Factual, and Counting/Fluency. Based on these defined classes, *Algorithm 1: Algorithm to classify QCoC classes* is executed with the original CoQA distribution as a point of reference. The output from this algorithm is a one-hot encoding value for each data point in the QCoC dataset, corresponding to the class for the question text in the data point. The overall result from the QCoC classification process is outlined in the following Table 4.2

Table 4.2 QCoC classification result

Item	Result
Total data point	116630
Class distribution	Yes/No: 22478 data points (19.27%) Unknown: 1450 data points (1.24%) Picking: 749 data points (0.64%) Factual: 64589 data points (55.38%) Counting/Fluency: 27364 (23.47%)

Referring to Table 4.2, the total number of data points for QCoC is 116,630. This total comprises the combined data points from the CoQA training and evaluation datasets. As mentioned in the previous Chapter 3, the classification of the Yes/No, Unknown, and Picking classes is relatively straightforward through the proposed Algorithm 1. However, the Factual and Counting/Fluency classes require additional processing involving dot product using cosine similarity calculation. Ultimately, the algorithm classifies the Factual class to represent 55.38% of all QCoC data points, while Counting/Fluency accounts for 23.47%. This classification is in accordance with the original CoQA distribution reported by the author (Reddy et al., 2019). To further illustrate the correlation between QCoC and CoQA distributions, Table 4.3 showcases the mapping between the distribution of QCoC and CoQA.

Table 4.3 QCoC classification result against CoQA

QCoC	CoQA
Yes/No, Picking, Factual and Counting/Fluency (98.76%)	Answerable (98.7%)
Unknown (1.24%)	Unanswerable (1.3%)
Yes/No (19.27%)	Yes and No (19.8%)
Factual (55.38%)	Named Entity, Noun Phrase, and Date/Time (52.2%)
Unknown, Picking and Counting/Fluency (25.35%)	Number and Other (27.9%)

Referring to Table 4.3, four out of the five QCoC classes are mapped with the CoQA Answerable type questions, with fairly similar percentages (98.76% for QCoC and 98.7% for CoQA). A closer examination of the distribution reveals the following:

- For the QCoC Yes/No class, it is matched with CoQA Yes (11.1%) and No (8.7%) type answers, resulting in 19.27% for QCoC and 19.8% for CoQA. This represents a straightforward matching between the Yes/No class and the Yes and No type answers.

- Regarding the QCoC Factual class, it is matched with CoQA Named Entity (28.7%), Noun Phrase (19.6%), and Date/Time (3.9%) type answers, yielding 55.38% for QCoC and 52.2% for CoQA.

- Lastly, the QCoC Unknown (1.24%), Picking (0.64%), and Counting/Fluency (23.47%) classes are matched with CoQA Number (9.8%) and Other (18.1%) type answers, resulting in 25.35% for QCoC and 27.9% for CoQA. This matching accounts for CoQA Unknown and abstractive type answers.

By completing this phase, a new dataset for the question classification process, named Question Classification of CoQA (QCoC), has been developed. This dataset differs from previous works in two significant aspects:

ii. The class is based on the answer's features, rather than the answer's context.

iii. The dataset focuses on the conversational Question-Answering (QA) domain, rather than the direct Machine Reading Comprehension (MRC) domain.

The unique class structure of QCoC has been designed to tackle the challenges faced by QA systems dealing with abstractive answers. Among the five phenomena associated with abstractive answers (Yes/No, Coreference, Counting, Picking, and Fluency), QCoC delineates five corresponding classes: Yes/No, Unknown, Picking, Factual, and Counting/Fluency. The identification of these classes is rooted in the original CoQA distribution categories upon which this dataset is built. In essence, QCoC can be defined as a multiclass text classification dataset that accentuates variations in a question's context through its answer's features, diverging from the approach of directly classifying questions based on the context of the answer, as seen in previous works in this field.

With the completion of this dataset, the subsequent phase of this research methodology involves the development of a modified word vectors method. This method is essential for altering the raw LLM-generated vector of the question to represent the question's context without necessitating modifications, re-training, or fine-tuning of the LLM, which can be costly. Two main algorithms are proposed for this method: 1) The general word weighting algorithm, which generates scalar values to emphasise the importance of general/question words within the question text, and 2) The modified word vectors algorithm, which embeds the produced scalar values into the LLM-generated word vectors to highlight the general/question words. Overall, this method will serve as a preprocessing stage, taking the raw LLM-generated vector as input and producing a weighted vector of the same dimension length as output.

4.2.3 Phase 3: Modified Word Vectors Method Development

In this phase, a method for modifying the LLM-generated word vectors into weighted word vectors is developed. This method aims to assign desired weight targets to the input word vectors, particularly focusing on words that can differentiate question texts (i.e., the general/question words). Two activities are executed in this phase: activity one involves the development of a general word weighting algorithm, and activity two focuses on the development of a modified word vectors algorithm.

- **Activity 1**: Development of general word weighting algorithm

Two experiments are conducted to assess the proposed *Algorithm 2: Algorithm to generate word's scalar weight*. The first experiment, denoted "experiment A," exclusively utilizes the CoQA dataset, while the second experiment, "experiment B," incorporates a combination of CoQA, SQUAD, and QuAC datasets. These experiments aim to evaluate the scalability of Algorithm 2. The difference between the total tokens in Experiment A and Experiment B is 2,171,178, with a variance of 7,433 unique tokens.

Upon executing Algorithm 2 for both experiment A and B, a scalar weight value is produced for each unique token/word in the dataset. This weight value is represented as the TF percentage, calculated based on the total unique tokens present in the given dataset. To outline the key findings from both experiments, Table 4.4 showcases the overall results for these experiments.

Table 4.4 Overall results of general word weighting algorithm

Items	Experiment A	Experiment B
Dataset	CoQA	CoQA, SQUAD and QuAC
Total Token (TT)	662607	2833785
Total Unique Token (UT)	17404	24455
Maximum TF percentage	5.92%	5.55%
Minimum TF percentage	0.00015%	0.00018%

In Table 4.4, the total number of tokens (TT) for experiments A and B are 662,607 and 2,833,785, respectively. This indicates a scale-up of more than 300% for experiment B compared to experiment A. On the other hand, the difference in unique tokens (UT) is only 7,051 tokens, which represents a scale-up of around 40% for experiment B from experiment A. Furthermore, the maximum and minimum TF percentage values show only slight differences, with a margin of 0.37% for the maximum and 0.00003% for the minimum. For a more detailed analysis, the TF percentage values for each token are segmented into ten clusters. It is worth noting that a higher percentage value implies that the token is highly general or common, as justified by its frequent usage in the given dataset. Conversely, a lower percentage value suggests that the token is highly specific or uncommon, as justified by its limited usage in the dataset. The distribution of UT regarding the ten clusters is visually represented in the following Figure 4.1.

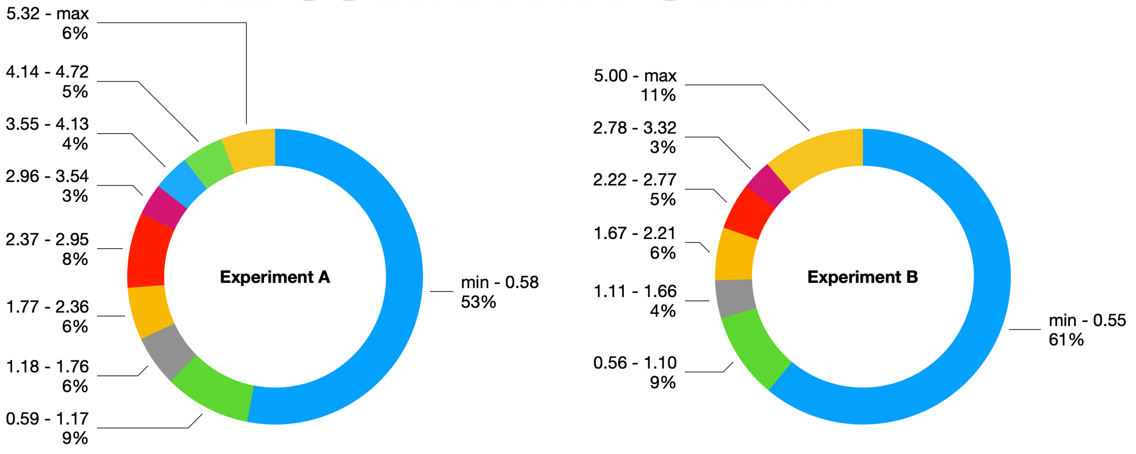


Figure 4.1 Distribution of Unique Token (UT) TF percentage

Referring to Figure 4.1, a significant portion of each experiment is dominated by the lowest cluster of TF percentage (53% for experiment A and 61% for experiment B). This observation suggests that the disparity in usage between general and specific words is considerable, as lower TF percentage values indicate lower token/word usage in the dataset. To conduct a more in-depth analysis of the ten segmented clusters, subsequent Tables 4.5 and 4.6 present the TF percentage clusters' range, total TF percentage for each cluster, TF count for each cluster, UT counts for each cluster and a *Sum* value in a specific column.

Table 4.5 Detail TF percentage distribution over ten clusters (experiment A)

Cluster	**Total % in cluster**	**TF count**	**UT count**	**Sum**
Min – 0.58	52.90	350,519	17,379	N/A
0.59 – 1.17	9.45	62,616	11	11
1.18 – 1.76	5.53	36,642	4	15
1.77 – 2.36	5.72	37,901	3	18
2.37 – 2.95	8.19	54,268	3	21
2.96 – 3.54	3.48	23,059	1	22
3.55 – 4.13	3.99	26,438	1	23
4.14 – 4.72	4.56	30,215	1	24
4.73 – 5.31	0.00	0	0	24
5.32 – Max	5.92	39,226	1	25
Total	**100**	**662,607**	**17,404**	**25**

Table 4.6 Detail TF percentage distribution over ten clusters (experiment B)

Cluster	**Total % in cluster**	**TF count**	**UT count**	**Sum**
Min – 0.55	60.54	1,715,573	24,430	N/A
0.56 – 1.10	9.36	265,242	14	14
1.11 – 1.66	4.11	116,469	3	17
1.67 – 2.21	5.74	162,659	3	20
2.22 – 2.77	5.08	143,956	2	22
2.78 – 3.32	3.26	92,381	1	23
3.33 – 3.88	0.00	0	0	23
3.89 – 4.43	0.00	0	0	23
4.44 – 4.99	0.00	0	0	23
5.00 – Max	11.11	314,834	2	25
Total	**100**	**2,833,785**	**24,455**	**25**

Referring to Tables 4.5 and 4.6, the *Sum* column is included to showcase that for both experiments, the sum of UT count for the nine highest clusters is the same (25 UT count, as shown in the Total row for the Sum column). Despite the vast difference in total TT and UT count for each experiment (with a difference of 2,171,178 TT and 7,433 UT count between experiments A and B), this result indicates that the total number of majorly used words is similar. It suggests that although the dataset is scaled up (with around 300% more text/tokens), the amount of unique general/question words that are used remains immensely similar. Upon examining this claim, the following are the 25 UT words for each experiment, listed in sequence from highest to lowest TF percentage:

- **Experiment A**: [what] [the] [did] [was] [he] [who] [is] [to] [how] [it] [of] [in] [they] [where] [a] [does] [<u>she</u>] [do] [when] [<u>his</u>] [for] [<u>many</u>] [that] [s] [were]
- **Experiment B**: [the] [what] [did] [of] [was] [in] [is] [to] [he] [who] [how] [a] [when] [for] [s] [are] [do] [it] [where] [they] [does] [were] [<u>any</u>] [<u>and</u>] [<u>that</u>]

From the listed 25 UT words for both experiments, only three words are not similar between the two sets (denoted by underlined text in the list). Although not 100% similar, this result shows that common general word usage is highly similar despite being measured from different dictionaries or document sizes. This finding further supports the generalizability of the proposed algorithm, indicating its ability to consistently capture and emphasise common general/question words, regardless of the dataset's scale or size.

With the completion of this activity, a list of token-weight pairs is produced. The weight value in this pair corresponds to the TF percentage value (a scalar value), which signifies the weightage of its paired token. Notably, from both experiments, the maximum TF percentage value does not exceed a single-digit integer, indicating that the TF percentage is expected to effectively emphasise the weightage of general words without overshadowing the overall contextual semantics of the given text. To further validate this expectation, activity two in this phase involves developing an algorithm to embed this scalar value into the raw LLM-generated word vectors, and analysis regarding its contextual representation is performed.

- **Activity 2**: Development of modified word vectors algorithm

Two comparison experiments are conducted to assess the proposed '*Algorithm to generate weighted fixed-length vectors*.' Experiment one involves sentence similarity measurement, and experiment two focuses on vector differences measurement. For both experiments, five test cases are compared. Additionally, two types of word vectors are calculated for both experiments: those with the g-weight implementation (type A with four variations: A-V1, A-V2, A-V3, and A-V4) and those without the g-weight implementation (type B). The four variations for type A are as follows: Algorithm 3 for variation 1 (A-V1), Algorithm 4 (A-V2), Algorithm 5 (A-V3), and Algorithm 6 (A-V4). For type A (A-V1 to A-V4), the produced word vectors are derived from the four algorithms presented in Chapter 3, while for type B, the produced word vectors are the original outputs from USE LLM. The following subsections will present the results of the experiments, starting with experiment one and followed by experiment two.

Experiment 1: Sentence Similarity Measurement

For the sentence similarity measurement, the dot product is calculated for each test case against each other. The results for type A (g-weight implementation) and type B (original USE LLM word vectors) are presented in Tables 4.7 to 4.11, respectively. It should be noted that some values are high in decimal-point and require a scalable factor to produce values that are readable and suitable for the presented table formats. A caption indicating this has been included in the relevant tables.

Table 4.7 Dot product between test cases for type A-V1 (g-weight)

No	**1**	**2**	**3**	**4**	**5**
1	0.006 (5)	0.014 (5)	1.799 (5)	2.747 (5)	0.045 (5)
2	0.014 (4)	0.051 (4)	4.411 (4)	6.717 (4)	0.115 (4)
3	1.799 (2)	4.412 (2)	595.904 (2)	916.585 (2)	14.722 (2)
4	2.747 (1)	6.717 (1)	916.585 (1)	1425.34 (1)	22.481 (1)
5	0.045 (3)	0.115 (3)	14.722 (3)	22.481 (3)	0.369 (3)

*Result is multiplied by 100 to scale into a readable format

Table 4.8 Dot product between test cases for type A-V2 (g-weight)

No	1	2	3	4	5
1	0.00001 (5)	0.0001 (5)	0.054 (5)	0.664 (5)	0.0001 (5)
2	0.00017 (3)	0.0016 (3)	0.475 (3)	5.993 (3)	0.0014 (3)
3	0.054 (2)	0.475 (2)	157.328 (2)	1939.72 (2)	0.428 (2)
4	0.664 (1)	5.993 (1)	1939.72 (1)	25392.6 (1)	5.457 (1)
5	0.00016 (4)	0.0014 (4)	0.428 (4)	5.457 (4)	0.0013 (4)

*Result is multiplied by 100000 to scale into a readable format

Table 4.9 Dot product between test cases for type A-V3 (g-weight)

No	1	2	3	4	5
1	123.714 (4)	169.746 (2)	127.207 (4)	118.623 (4)	109.117 (3)
2	169.747 (1)	368.139 (1)	167.080 (1)	162.402 (1)	132.942 (1)
3	127.207 (2)	167.080 (3)	154.438 (2)	134.677 (2)	106.047 (5)
4	118.623 (3)	162.402 (4)	134.677 (3)	130.702 (3)	107.309 (4)
5	109.117 (5)	132.942 (5)	106.047 (5)	107.309 (5)	114.366 (2)

Table 4.10 Dot product between test cases for type A-V4 (g-weight)

No	1	2	3	4	5
1	32.405 (5)	77.871 (5)	46.595 (5)	65.427 (5)	74.312 (5)
2	77.871 (1)	189.435 (1)	112.224 (1)	158.306 (1)	178.094 (1)
3	46.595 (4)	112.224 (4)	67.325 (4)	94.757 (4)	107.080 (4)
4	65.427 (3)	158.306 (3)	94.757 (3)	135.203 (3)	151.243 (3)
5	74.312 (2)	178.094 (2)	107.080 (2)	151.243 (2)	172.938 (2)

Table 4.11 Dot product between test cases for type B (original USE)

No	1	2	3	4	5
1	125.440 (1)	123.421 (3)	121.252 (3)	117.420 (5)	122.683 (3)
2	123.421 (2)	125.440 (1)	119.181 (5)	118.338 (4)	124.132 (2)
3	121.252 (4)	119.181 (4)	125.440 (1)	121.736 (2)	119.504 (5)
4	117.420 (5)	118.338 (5)	121.736 (2)	125.440 (1)	119.982 (4)
5	122.683 (3)	124.132 (2)	119.504 (4)	119.982 (3)	125.440 (1)

The previously presented Tables 4.7 to 4.11 offer the results from the dot product values obtained from the sentence similarity comparison process for both type of word vectors: those with and without g-weight implementation. Each table includes rankings denoted by numbers in brackets, indicating the level of semantic similarity between pairs of questions. Higher dot product values signify closer semantic similarity, while lower values indicate greater dissimilarity. Notably, Table 4.11, which represents the original USE LLM word vectors without g-weight implementation, consistently shows the first ranking to be when a sentence is compared to itself. This is expected because the magnitude of USE LLM word vectors lies within the same dimensional radius, leading to dot products not exceeding the value obtained when the sentence is compared to itself (due to the dot product of a vector with itself being the squared magnitude of the vector). On the other hand, the introduction of g-weight as an external value towards the original USE LLM word vectors significantly alters the dimensional radius for each sentence based on its associated g-weight value.

Upon observation of the g-weight implementation cases (Tables 4.7, 4.8, and 4.10), the dot product values exhibit a noticeable pattern, consistently increasing in parallel with the corresponding g-weight value, except for type A-V3 (Table 4.9). Despite this anomaly in variation 3, the rankings for all test cases tend to be relatively identical, even when a question is compared against itself. This suggests that the g-weight implementation contributes to the increased dot product values and reinforces the semantic similarity between questions, aligning with the intended aims of this algorithm, which is to modify the LLM-generated word vectors to emphasise general words as the external weight factor for the question texts.

To provide a different perspective on the findings, Table 4.12 presents the rankings for the five test cases in type A variations A-V1, A-V2, and A-V4, along with their corresponding general words and total g-weight values (the sum of all g-weight values in each test case). Variation A-V3 is excluded from the table as this specific algorithm doesn't produce the expected result, hence being concluded as ineffective in emphasising the general words.

Table 4.12 The ranking of the four variations in type A with total g-weight values

No	Question text	General words	Total g-weight
1	how many items are on sale? A-V1 (5), A-V2 (5) and A-V4 (5)	how, many, are, on	2.909
2	what items are on sale? A-V1 (4), A-V2 (3) and A-V4 (1)	what, are, on	6.731
3	how many people is allowed? A-V1 (2), A-V2 (2) and A-V4 (4)	how, many, is	3.515
4	who is allowed? A-V1 (1), A-V2 (1) and A-V4 (3)	who, is	3.132
5	where are the items for sale? A-V1 (3), A-V2 (4) and A-V4 (2)	where, are, the, for	7.578

Referring to Table 4.12, it can be observed that variation A-V4 appears to closely approximate the correct ranking, showing a tendency to ascend towards higher total g-weight values. However, it should be noted that the ranking is not entirely precise, as expected due to the presence of both positive and negative numbers in the word vectors. To elaborate, multiplying vectors with a scalar (g-weight value) may amplify negative values, resulting in larger negative numbers, and vice versa. Despite this imprecision, the overall perspective of variation A-V4's results can be viewed favorably in justifying the significance of general words after being weighted using the proposed algorithm.

In contrast, for type B (original USE embeddings without g-weight implementation), the sentence similarity favours context words, as presented in Table 4.11. Without considering general words, questions containing the words ‘items’ and ‘sale’ are calculated to be similar to each other, and questions with the word ‘allowed’ are also found to be similar to one another. This aligns with the main purpose of USE or any other LLM, which is to emphasise context words.

To summarise the results from both type A and B, it can be concluded that with the correct algorithm, the g-weight implementation can impact the original LLM-generated word vectors, specifically in emphasising the usage of general words. In broader generalisation, this finding supports the hypothesis that an external scalar weight value can be used to modify or alter the original LLM-generated word vectors without modifying, retraining, or fine-tuning the pretrained LLM.

Experiment 2: Vector Differences Measurement

To further analyse the significance of the g-weight implementation's impact on the original LLM-generated word vectors, a comparison of vector value differences is conducted for similar test cases and word vector types (A and B). As previously mentioned, the vector difference is calculated using direct matrix subtraction, where the first matrix is subtracted from the second matrix, and then the sum of all differences is computed to produce the final reported value. This raw vector value is crucial because it will serve as the input features for the vanilla feed-forward Neural Network (FF-NN) model in the next phase of this research methodology. Consequently, the most similar input features will be classified as similar, and vice versa. The results of this vector value difference comparison for type A (g-weight implementation) and type B (original USE embeddings) are presented in Tables 4.13 to 4.17.

Table 4.13 Vector value difference for type A-V1 (g-weight)

No	**1**	**2**	**3**	**4**	**5**
1	0 (1)	0. 057 (2)	5.368 (5)	8.191 (5)	0.128 (3)
2	0.057 (2)	0 (1)	5.312 (4)	8.134 (4)	0.071 (2)
3	5.368 (4)	5.312 (4)	0 (1)	2.822 (2)	5.240 (4)
4	8.191 (5)	8.134 (5)	2.822 (2)	0 (1)	8.062 (5)
5	0.128 (3)	0.071 (3)	5.240 (3)	8.063 (3)	0 (1)

Table 4.14 Vector value difference for type A-V2 (g-weight)

No	**1**	**2**	**3**	**4**	**5**
1	0 (1)	0.00023 (3)	0.08909 (4)	1.027 (5)	0.00021 (3)
2	0.00023 (3)	0 (1)	0.08886 (2)	1.02641 (3)	0.00001 (2)
3	0.089 (4)	0.0889 (4)	0 (1)	0.938 (2)	0.088 (4)
4	1.027 (5)	1.026 (5)	0.938 (5)	0 (1)	1.026 (5)
5	0.00021 (2)	0.00001 (2)	0.08888 (3)	1.02644 (4)	0 (1)

Table 4.15 Vector value difference for type A-V3 (g-weight)

No	1	2	3	4	5
1	0 (1)	11.339 (4)	3.970 (2)	0.0443 (2)	2.825 (3)
2	11.339 (5)	0 (1)	7.369 (5)	11.384 (5)	14.164 (5)
3	3.970 (4)	7.369 (2)	0 (1)	4.015 (4)	6.795 (4)
4	0.044 (2)	11.384 (3)	4.015 (3)	0 (1)	2.780 (2)
5	2.825 (3)	14.164 (5)	6.795 (4)	2.780 (3)	0 (1)

Table 4.16 Vector value difference for type A-V4 (g-weight)

No	1	2	3	4	5
1	0 (1)	12.981 (5)	4.402 (2)	13.916 (5)	13.787 (5)
2	12.981 (3)	0 (1)	8.579 (3)	0.935 (3)	0.806 (3)
3	4.402 (2)	8.579 (4)	0 (1)	9.514 (4)	9.385 (4)
4	13.916 (5)	0.935 (3)	9.514 (5)	0 (1)	0.130 (2)
5	13.787 (4)	0.806 (2)	9.385 (4)	0.130 (2)	0 (1)

Table 4.17 Vector value difference for type B (original USE)

No	1	2	3	4	5
1	0 (1)	1.892 (5)	0.199 (2)	2.548 (5)	0.975 (4)
2	1.892 (4)	0 (1)	1.692 (4)	0.656 (2)	0.916 (3)
3	0.199 (2)	1.692 (4)	0 (1)	2.348 (4)	0.776 (2)
4	2.548 (5)	0.656 (2)	2.348 (5)	0 (1)	1.572 (5)
5	0.975 (3)	0.916 (3)	0.776 (3)	1.572 (3)	0 (1)

Tables 4.13 to 4.17 provide the vector value differences (in real numbers) for both type of word vectors: those with and without g-weight implementation. Similar to previous comparison tables, the rankings (smallest to highest difference) are included in the tables. The goal of this vector value comparison is to obtain smaller differences between vectors of similar question types, indicating closer similarity between their vector values. As these vectors are intended to serve as input features for question type classification ML model, a smaller difference signifies a higher level of similarity between the vectors of questions of the same type.

Upon examining the five test cases, it is expected that the vector for question number (1) “How many items are on sale?” should closely resemble the vector for question number (3) “How many people is allowed?” (both are ‘how many’ type questions). Similarly, the vectors for question number “What items are on sale?”, (4) “Who is allowed?”, and (5) “Where are the items for sale?” should all closely resemble each other, as they are factual-type questions. The smaller the differences between these vector pairs, the more effectively the proposed method is emphasising general words and reinforcing the semantic similarity between questions of the same type.

Referring to Tables 4.13 to 4.17, only Table 4.16 (type A-V4) and Table 4.17 (type B) satisfy the requirement that “question number (1) and (3) should be closely similar to one another”. Upon further comparison of these two tables, question number (1) and (3) are ranked last and second last (rank 4 and 5) in the type A-V4 result. In contrast, type B’s result is also close, but it mistakenly ranks question number (5) closely similar to question number (3). The favorable outcome in type A-V4 is attributed to the fact that questions (2), (4), and (5) fall under the factual/extractive type questions (“what,” “who,” and “where” questions), while question (1) and (3) belong to the abstractive type questions (“how many” questions, which fall under the *Counting* phenomenon). This shows that the proposed method, particularly in variation 4 (A-V4), effectively emphasises general words by embedding their scalar weight value into the original LLM-generated word vectors. As a result, the modified word vectors produced by the algorihtm reinforce the similarity between questions of the same types.

Further examining the values in Table 4.16 (type A-V4), it becomes evident that there is a substantial gap between the abstractive and factual question types. This gap can be identified by analysing each difference value, whereby a significant difference is observed when comparing questions with different types. For example, in column 1 for question 1, the difference between this question and questions 2, 4, and 5 (which are different in type) is relatively higher than between question 3 (which is the same type). To further illustrate this gap, Figures 4.2 and 4.3 visualize the vector values in 3-dimensional line graph format for both type A-V4 and type B, respectively.

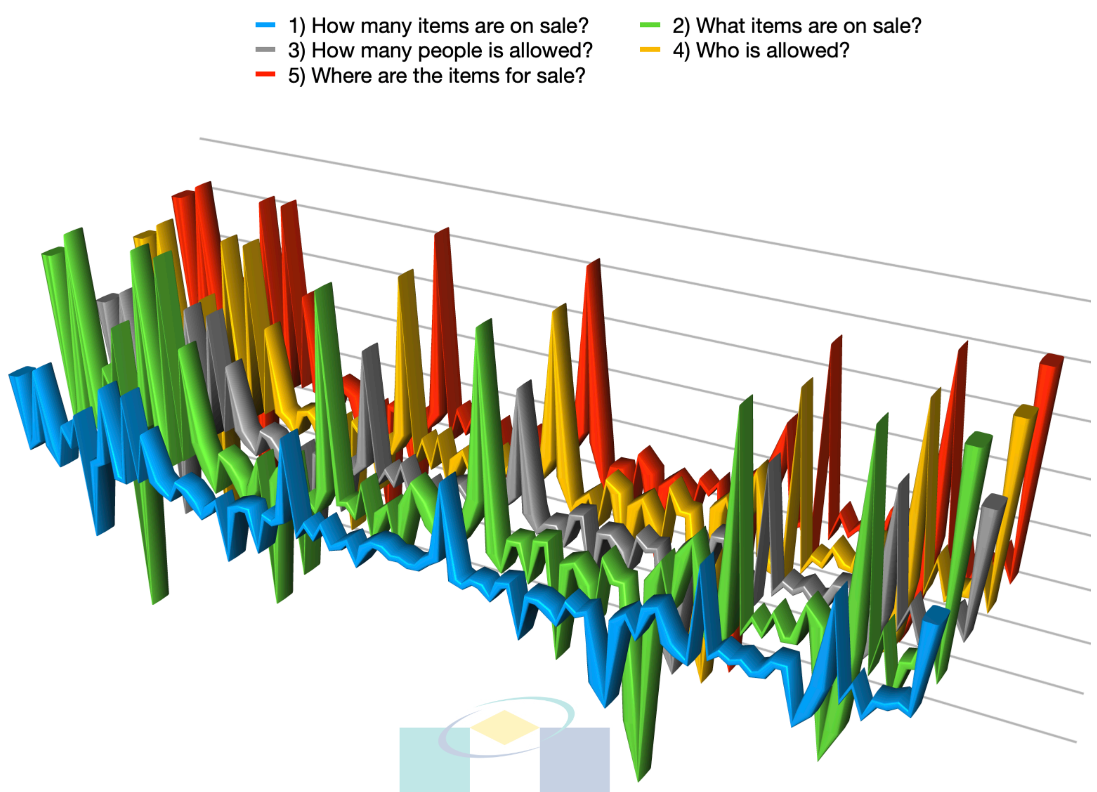


Figure 4.2 Vector graph for type A-V4 (g-weight implementation)

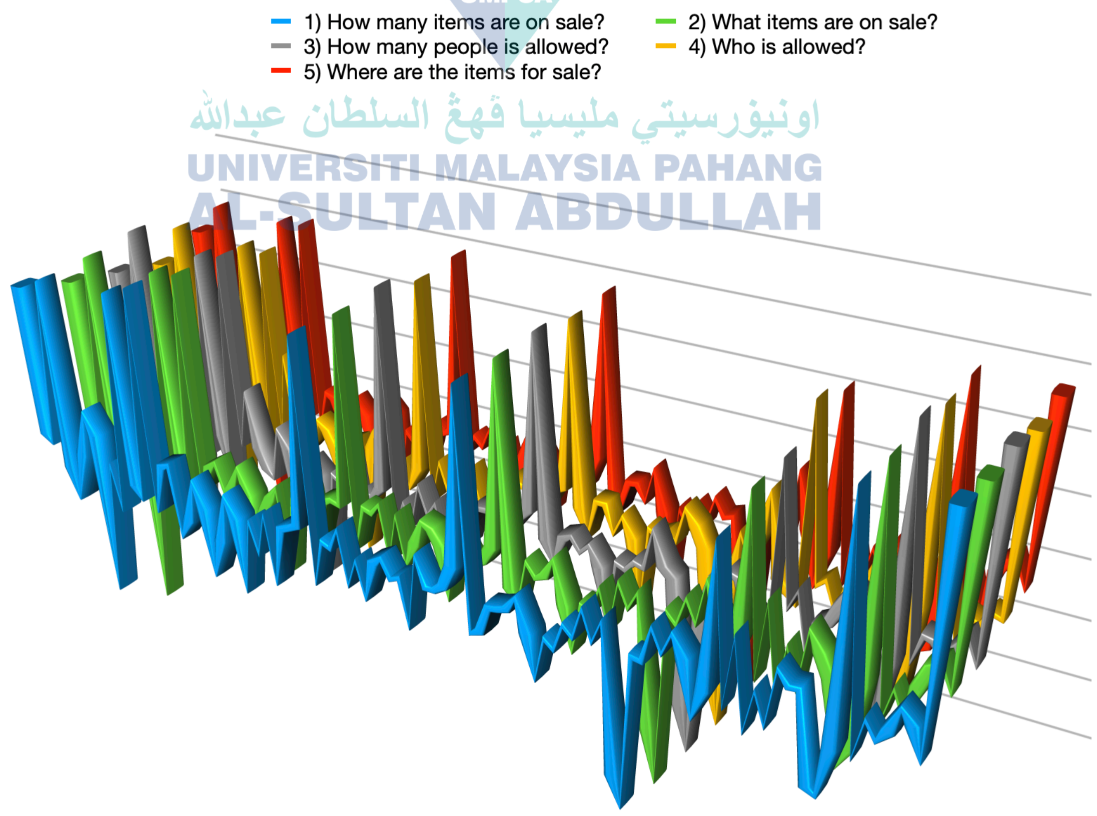


Figure 4.3 Vector graph for type B (original USE embedding)

The previous Figures 4.2 and 4.3 visually represent the 100-dimensional vector values for all test cases in both type A-V4 and type B using 3-dimensional line graphs. A bird's-eye view comparison of both figures reveals that type A-V4 (with g-weight implementation) exhibits a noticeable difference between the vectors, whereas type B (original USE embedding without g-weight implementation) shows relatively little difference. Specifically, the difference in type A-V4 is depicted by the blue and grey lines, which exhibit dissimilar spikes compared to the other lines. This difference aligns with the aim of this proposed method, which is to generate a weighted fixed-length vector capable of representing question-type features using both general and question words. As previously mentioned, the question number (1) "How many items are on sale?" vector should closely resemble question number (3) "How many people are allowed?" vector (both being 'how many' type questions). In the presented Figure 4.2 for type A-V4, this similarity is evident, with the blue line representing question number (1) and the grey line representing question number (3) showing a close resemblance to each other compared to other question lines.

This Phase 3 has showcased the successful method of modifying LLM-generated word vectors using an external scalar weight value. This method was achieved by implementing two algorithms, namely: 1) General weight weighting algorithm and 2) Modified word vectors algorithm. For the second algorithm, four variations are proposed, namely A-V1, A-V2, A-V3, and A-V4. Based on the previously discussed results, variation A-V4 has showcased the best result, which aligns with the overall aims of this proposed modified word vectors method. In summary, this method implements Algorithm 2 and Algorithm 6 (A-V4) to achieve the intended result.

With the completion of this phase, the next phase of this research methodology will involve the development of a machine learning classifier using a vanilla/base feed-forward Neural Network (FF-NN) algorithm for the question-type classification task of the QCoC dataset, developed in Phase 2 of this research. The developed modified word vectors method in this phase will serve as a preprocessing module before the word vectors data are used to train and evaluate the machine learning classifier. In the overall flow, the pre-trained LLM will be used to generate raw contextualised word vectors for QCoC datapoints, which will then undergo modification using Algorithm 6 before being fed into the machine learning classifier for further analysis.

4.2.4 Phase 4: Machine Learning Classifier Development

In this phase, eight sequential experiments are conducted to develop a Machine Learning (ML) classifier for the question-type classification task of the QCoC dataset. As mentioned, this ML classifier will be built using a vanilla/base feed-forward Neural Network (FF-NN) algorithm, aiming for the most economical (least computationally intensive) model suitable for multiclass classification problems. The eight sequential experiments are designed and conducted to gradually build the ML classifier, starting from the most basic setup (minimum parameter count) and progressing towards the most advanced setup (maximum parameter count), in an attempt to achieve the best balance of economy and performance. To recap, the eight sequential experiments are as follows:

i. Determining hidden layer node counts for one hidden layer
ii. Determining the best combination of hidden layer and epoch size
iii. Determining the best activation function
iv. Determining the best optimizer function
v. Determining the best combination of batch and epoch size
vi. Determining the best loss function
vii. Determining the best combination of the loss function and epoch size
viii. Determining the best learning rate

Following the execution of these experiments, Table 4.18 presents the results.

Table 4.18 Results for the eight sequential experiments process

No	Setup	Result
1	Determining hidden layer node counts for one hidden layer *Default setup: ELU activation, Adamax optimizer, Categorical Cross Entropy loss, 10 epochs, 10% batch size (11518)*	
	i. 52 nodes (mean for input + output size)	F1 = 0.606
	ii. 100 nodes (input size)	F1 = **<u>0.662</u>**
2	Determining the best combination of hidden layer and epoch size *Default setup: ELU activation, Adamax optimizer, Categorical Cross Entropy loss, 10% batch size (11518)*	

i.	One hidden layer (100 nodes), 10 epochs	F1 = 0.662
ii.	One hidden layer (100 nodes), 50 epochs	F1 = 0.706
iii.	One hidden layer (100 nodes), 100 epochs	F1 = 0.710
iv.	One hidden layer (150 nodes), 10 epochs	F1 = 0.633
v.	Two hidden layers (100 nodes each), 10 epochs	F1 = 0.668
vi.	Two hidden layers (100 nodes each), 50 epochs	F1 = 0.714
vii.	Two hidden layers (100 nodes each), 100 epochs	F1 = 0.720
viii.	Two hidden layers (100 nodes each), 150 epochs	F1 = 0.723
ix.	Three hidden layers (100 nodes each), 10 epochs	F1 = 0.698
x.	Three hidden layers (100 nodes each), 50 epochs	F1 = 0.718
xi.	Three hidden layers (100 nodes each), 100 epochs	F1 = 0.723
xii.	Four hidden layers (100 nodes each), 50 epochs	F1 = 0.721
xiii.	Four hidden layers (100 nodes each), 100 epochs	F1 = 0.723
xiv.	Five hidden layers (100 nodes each), 100 epochs	F1 = **0.726**
xv.	Six hidden layers (100 nodes each), 100 epochs	F1 = 72.517

3 Determining the best activation function

Default setup: Adamax optimizer, Categorical Cross Entropy loss, 10% batch size (11518), five hidden layers (100 nodes each), 100 epochs

i.	ELU	F1 = 0.726 273829ms
ii.	ReLU	F1 = **0.726** **187079ms**
iii.	Leaky ReLU	F1 = 0.725 294017ms
iv.	PReLU	F1 = 0.727 194507ms
v.	Linear 5	F1 = 0.724 162263ms

4 Determining the best optimizer function

Default setup: Categorical Cross Entropy loss, 10% batch size (11518), five hidden layers (100 nodes each), 100 epochs, ReLU activation

	i.	Adam	F1 = 0.725 280892ms
	ii.	Adamax	F1 = **0.726** **187079ms**
	iii.	Momentum (0.001 learning rate)	F1 = 0.701 272572ms
	iv.	RMSProp	F1 = 0.675 235543ms
	v.	SGD (0.001 learning rate)	F1 = 0.561 240339ms
5		Determining the best combination of batch and epoch size *Default setup: Categorical Cross Entropy loss, five hidden layers (100 nodes each), ReLU activation, Adamax optimizer*	
	i.	1% batch size (1151), 10 epochs	F1 = 0.720 138268ms
	ii.	1% batch size (1151), 20 epochs	F1 = 0.726 354546ms
	iii.	1% batch size (1151), more than 20 epochs	*System couldn't handle*
	iv.	10% batch size (11518), 100 epochs	F1 = **0.726** **187079ms**
6		Determining the best loss function *Default setup: five hidden layers (100 nodes each), ReLU activation, Adamax optimizer, 10% batch size (11518), 100 epochs*	
	i.	Categorical Cross Entropy	F1 = **0.726** **187079ms**
	ii.	Cosine Distance	F1 = 56.077 285750ms
	iii.	Hinge Loss	F1 = 56.077 218393ms
	iv.	Mean Squared Error	F1 = **0.726** **170880ms**

	v. Sigmoid Cross Entropy	F1 = 70.735 185137ms
	vi. Softmax Cross Entropy	F1 = 72.209 195235ms
7	Determining the best combination of the loss function and epoch size *Default setup: five hidden layers (100 nodes each), ReLU activation, Adamax optimizer, 10% batch size (11518)*	
	i. Categorical Cross Entropy loss, 50 epochs	F1 = 0.724 77308ms
	ii. Categorical Cross Entropy loss, 70 epochs	F1 = **0.725** **155977ms**
	iii. Categorical Cross Entropy loss, 100 epochs	F1 = **0.726** **187079ms**
	iv. Mean Squared Error loss, 50 epochs	F1 = 0.723 75415ms
	v. Mean Squared Error loss, 70 epochs	F1 = 0.725 1522118ms
	vi. Mean Squared Error loss, 100 epochs	F1 = 0.726 170880ms
8	Determining the best learning rate *Default setup: five hidden layers (100 nodes each), ReLU activation, Adamax optimizer, 10% batch size (11518), Categorical Cross Entropy loss, 70 epochs*	
	i. Adamax 0.1 learning rate	F1 = 0.561 112125ms
	ii. Adamax 0.01 learning rate	F1 = **0.727** **111482ms**
	iii. Adamax 0.001 learning rate	F1 = 0.725 155977ms
	iv. Adamax 0.0001 learning rate	F1 = 0.682 111377ms

*ms = miliseconds

Table 4.18 presents the results for various FF-NN setups in this phase. A total of 47 NN setups have been tested, and from this sequential experiment, the highest-scoring F1 score is achieved using the following setup: five hidden layers, each with 100 nodes, ReLU activation function for all nodes in the input and hidden layers, a batch size of 10% or 11518 datapoints (mini-batch gradient descent), Adamax optimizer function with a learning rate of 0.01 for reducing loss generated by the Categorical Cross Entropy function, and 70 training epochs. Using this setup, an F1 score of 0.727 is achieved (approximately 30% higher than the baseline score) in just 111482 milliseconds (1 minute and 52 seconds) of training time using the hardware setup specified in Table 3.4.

From an economic perspective, the total parameter count of the produced FF-NN ML classifier model is only 51,604. This count is achieved through five hidden layers, each consisting of 100 nodes' weights and biases (10,200 parameters for one hidden layer), and one output layer with 100 nodes' weights and biases multiplied by 4 output nodes (404 parameters). The reduction in the number of training parameters is substantial compared to state-of-the-art methodology, which involves fine-tuning a Large Language Model (LLM). Fine-tuning LLM models for multiclass classification involves adding task-specific layers and may also include the base parameters of the pre-trained LLM for the training process. For reference, the parameter counts for various LLM models are as follows: 94 million for ELMo, 340 million for BERT (Large), 340 million for XLNET, 355 million for RoBERTa, 1.5 billion for GPT-2, 8.3 billion for Megatron-lm, 11 billion for T5, 17 billion for Turing-NLG, and a staggering 175 billion for GPT-3.

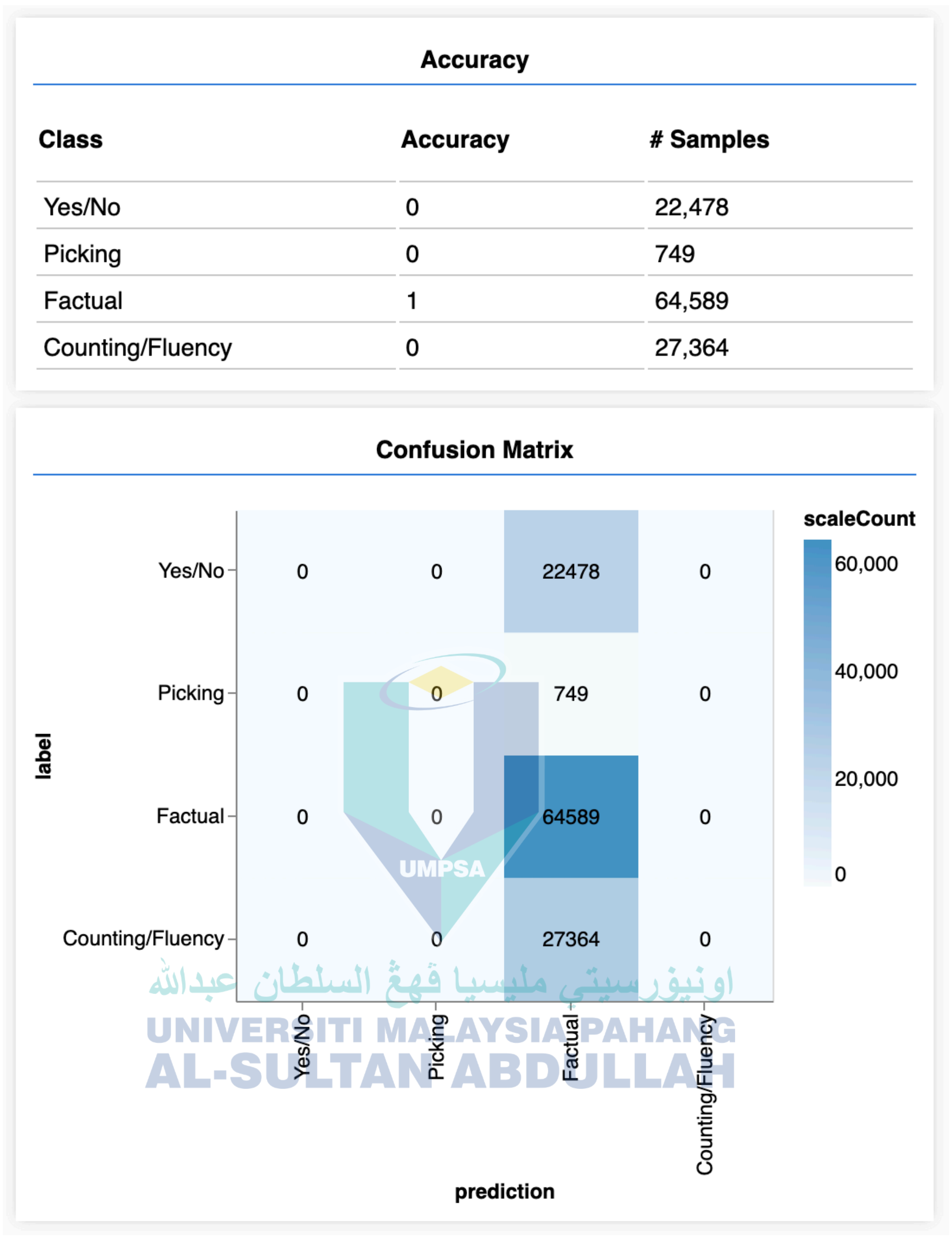
With the completion of this phase, the final step/phase in this research methodology is to validate and evaluate the ML classifier developed in this phase, incorporating additional evaluation metrics to analyse its correlation with the achieved F1 score. Specifically, the evaluation will concentrate on the classifier's capacity to predict classes for the QCoC dataset, with the aim of validating the research hypothesis: *the word representation vector derived from the LLM can be altered using an external scalar weight, which can later be used as input for an ML model to perform a text classification task.*

4.2.5 Phase 5: Methods Evaluation

In this final phase, all previously proposed methods will be collectively evaluated through the performance of the developed ML classifier on the QCoC dataset. Four evaluation metrics will be used: Accuracy, Precision, Recall, and F1 score. It is a well-known fact that NN models sometimes exhibit different results if not optimally trained. This is due to the nature of the NN training process, which assigns random weights to its nodes before starting the training process. Therefore, to ensure a comprehensive assessment, the classifier will undergo multiple testing iterations (three times each) to determine its stability and consistency.

To recap, the final classifier produced in the previous phase is a FF-NN model with the following setup: five hidden layers, each comprising 100 nodes, ReLU activation function applied to all nodes in the input and hidden layers, a 10% batch size (equivalent to 11518 datapoints) for mini-batch gradient descent, Adamax optimizer function with a learning rate of 0.01 to reduce loss generated by the Categorical Cross Entropy function, and 70 training epochs. As demonstrated in the previous phase, this particular setup achieved the highest F1 score, thus solidifying its status as the most optimal configuration for the classifier. However, in the interest of providing a comprehensive analysis, another setup will also be evaluated and reported. This alternative setup may exhibit higher generalization capabilities but result in a lower F1 score. The comparison of both setups will shed light on the trade-offs between generalization and performance, contributing to a more nuanced understanding of the model's behaviour.

As mentioned, the baseline accuracy for QCoC is 0.390, and the corresponding baseline F1 score is 0.561. This baseline value is obtained under the assumption that the classifier classifies all 115180 datapoints into the Factual class, which represents the highest number of data points in a single class (64589 out of 115180 or 56.08% of the overall dataset). Visual representation of the detailed accuracy per class and the confusion matrix for the baseline scenario is presented in the following Figure 4.4.



Accuracy

Class	Accuracy	# Samples
Yes/No	0	22,478
Picking	0	749
Factual	1	64,589
Counting/Fluency	0	27,364

Confusion Matrix

label \ prediction	Yes/No	Picking	Factual	Counting/Fluency
Yes/No	0	0	22478	0
Picking	0	0	749	0
Factual	0	0	64589	0
Counting/Fluency	0	0	27364	0

Figure 4.4 Baseline prediction for the four QCoC classes

Referring to Figure 4.4, the per-class accuracy for the *Factual* class is one (indicating the maximum accuracy), while the accuracy for other classes is zero (indicating the minimum accuracy). The confusion matrix displays the following values: true positives (TP) as 64589, false positives (FP) as 50591, and false negatives (FN) as 50591. Using these values, the baseline accuracy can be calculated using Equation 3.3 (accuracy), resulting in 0.390, and the baseline F1 score can be calculated using Equations 3.4 (precision), 3.5 (recall), and 3.6 (F1 score), resulting in 0.561. Table 4.19 outlines these baseline values.

Table 4.19 Baseline values for accuracy and F1 score

Metric	Baseline value
Accuracy (equation 3.3)	0.390
F1 score (equation 3.6)	0.561

Based on the baseline scores, two setups, namely Setup A and Setup B, are evaluated in this experiment. Setup A includes five hidden layers with 100 nodes in each layer, ReLU activation function for all nodes in the input and hidden layers, a batch size of 10% or 11518 data points for mini-batch gradient descent, Adamax optimizer function with a learning rate of 0.01, Categorical Cross Entropy loss function, and 70 training epochs. On the other hand, Setup B consists of three hidden layers with 100 nodes in each layer, ELU activation function for all nodes in the input and hidden layers, a batch size of 10% or 11518 data points for mini-batch gradient descent, Momentum optimizer function with a learning rate of 0.01, Categorical Cross Entropy loss function, and 10 training epochs. Table 4.20 outlines these setups parameters.

Table 4.20 Configuration parameters for setup A and setup B

Setup	Configuration parameters
A	5 hidden layers (100 nodes in each layer)
	ReLU activation function for all nodes (input and hidden layers)
	10% batch size (11518 data points for mini-batch gradient descent)
	Adamax optimizer function (0.01 learning rate)
	Categorical Cross Entropy loss function
	70 training epochs
B	3 hidden layers (100 nodes in each layer)
	ELU activation function for all nodes (input and hidden layers)
	10% batch size (11518 data points for mini-batch gradient descent)
	Momentum optimizer function (0.01 learning rate)
	Categorical Cross Entropy loss function
	10 training epochs

The training graphs for both setups are illustrated in following Figure 4.5 for Setup A and Figure 4.6 for Setup B, each showing three iterations. These graphs depict the training process and the model's performance during training for each iteration of the respective setups.

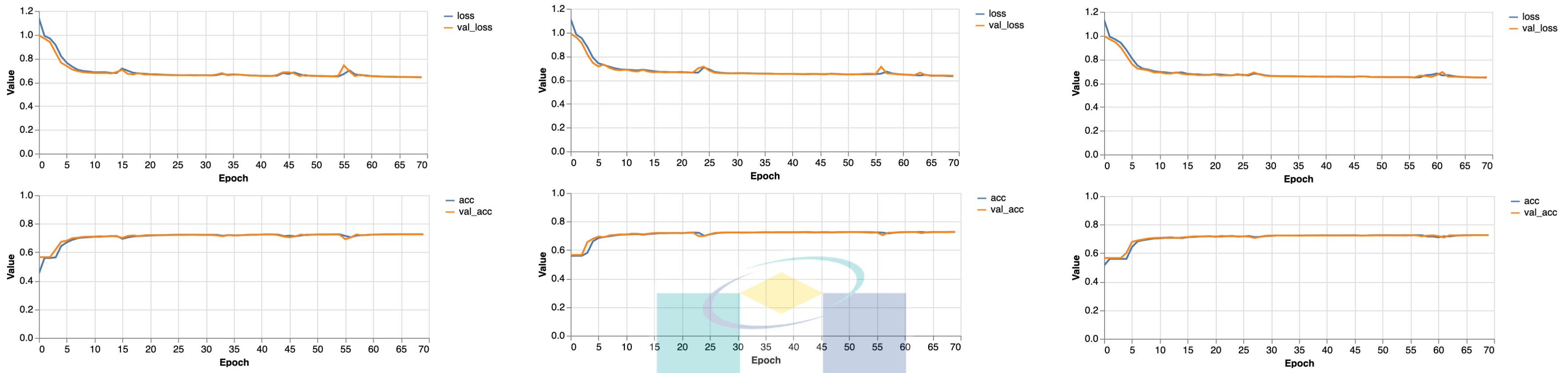


Figure 4.5 Training loss, validation loss, accuracy, and validation accuracy for setup A

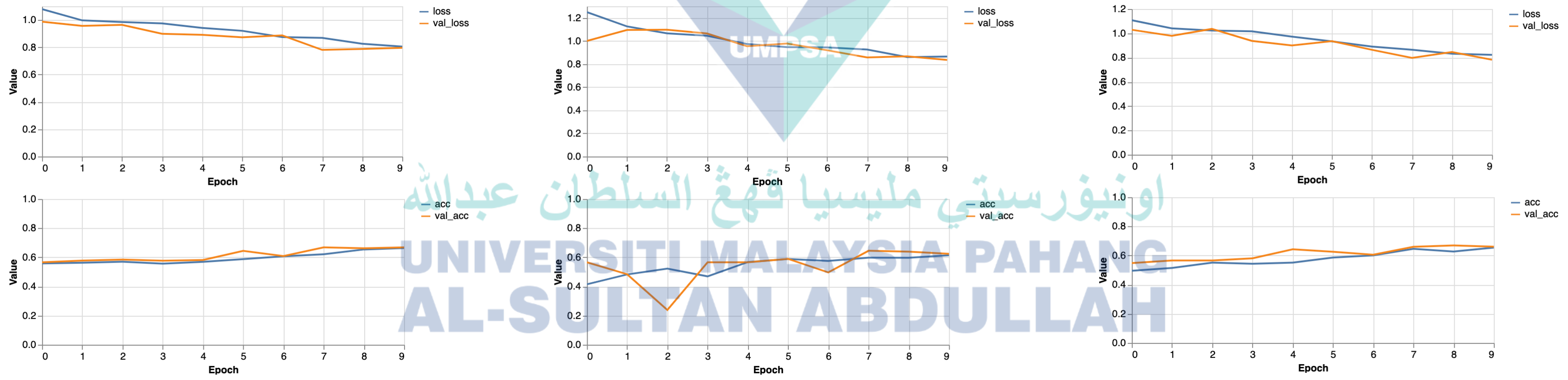
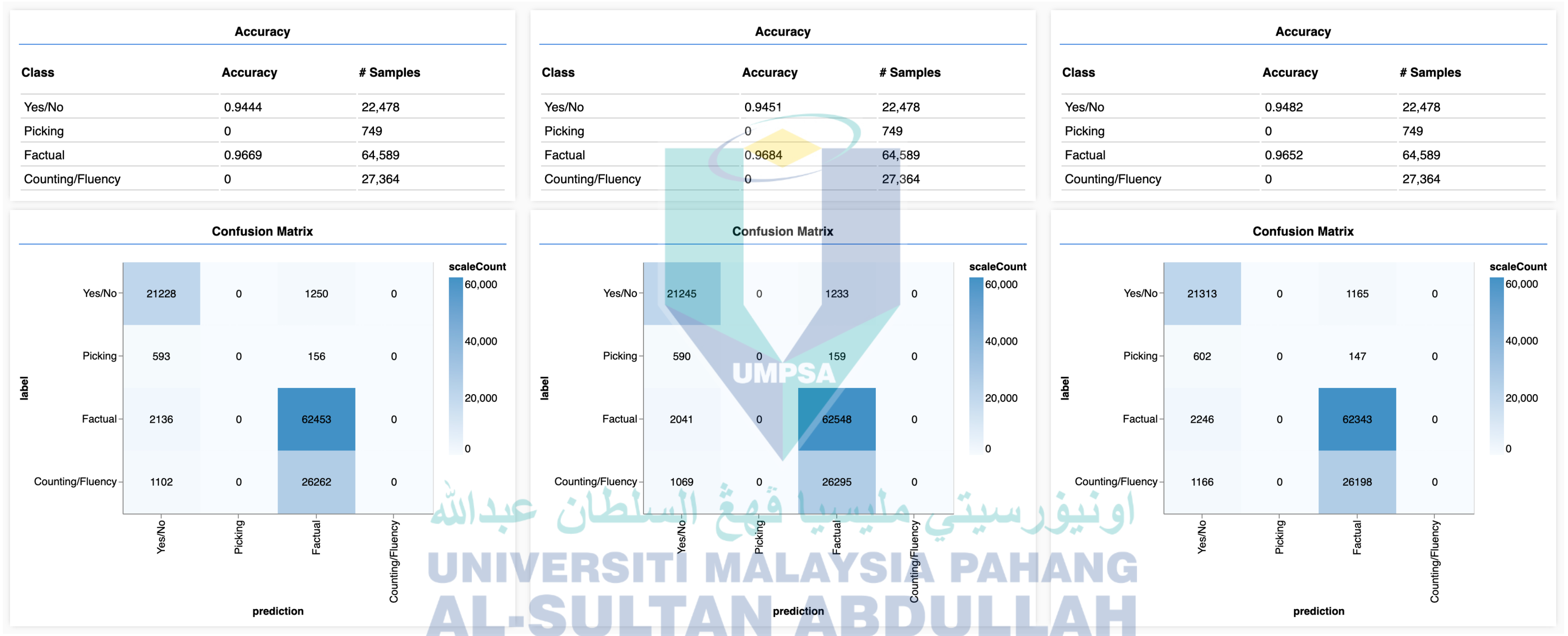


Figure 4.6 Training loss, validation loss, accuracy, and validation accuracy for setup B



Accuracy

Class	Accuracy	# Samples
Yes/No	0.9444	22,478
Picking	0	749
Factual	0.9669	64,589
Counting/Fluency	0	27,364

Confusion Matrix

label \ prediction	Yes/No	Picking	Factual	Counting/Fluency
Yes/No	21228	0	1250	0
Picking	593	0	156	0
Factual	2136	0	62453	0
Counting/Fluency	1102	0	26262	0

Accuracy

Class	Accuracy	# Samples
Yes/No	0.9451	22,478
Picking	0	749
Factual	0.9684	64,589
Counting/Fluency	0	27,364

Confusion Matrix

label \ prediction	Yes/No	Picking	Factual	Counting/Fluency
Yes/No	21245	0	1233	0
Picking	590	0	159	0
Factual	2041	0	62548	0
Counting/Fluency	1069	0	26295	0

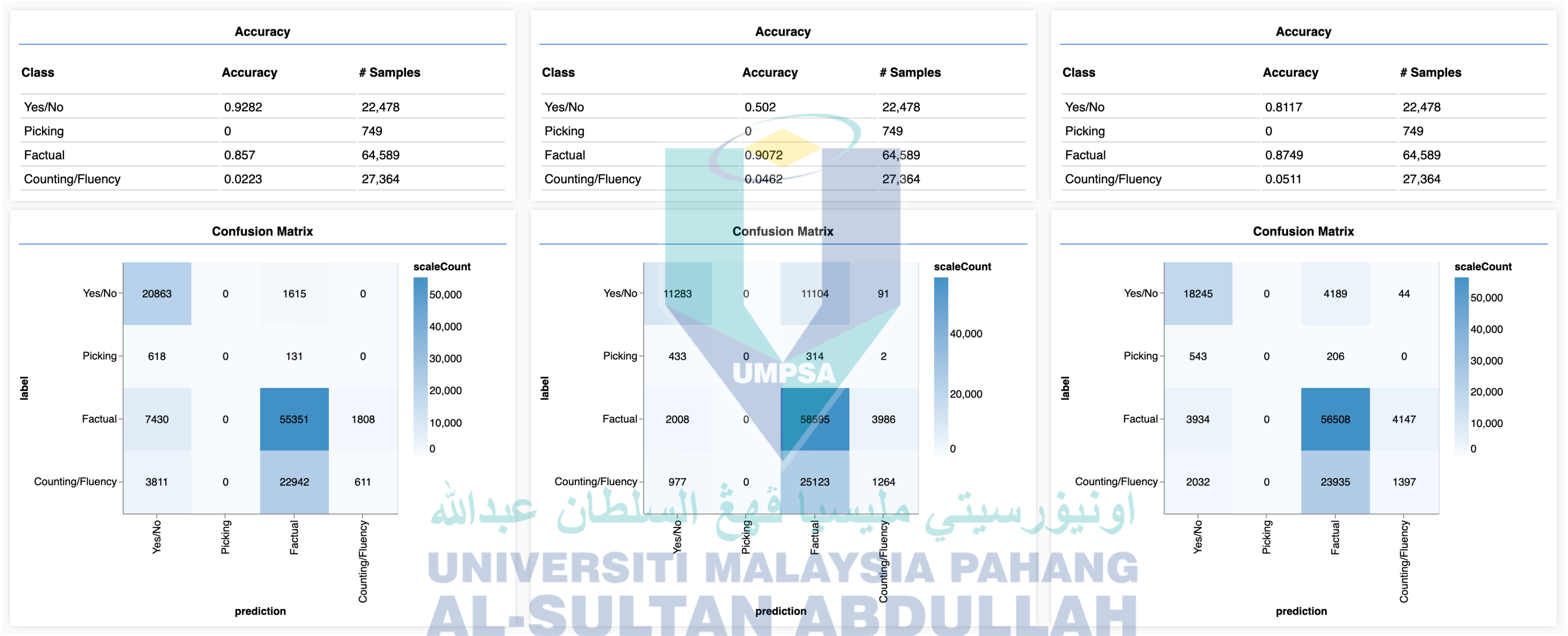
Accuracy

Class	Accuracy	# Samples
Yes/No	0.9482	22,478
Picking	0	749
Factual	0.9652	64,589
Counting/Fluency	0	27,364

Confusion Matrix

label \ prediction	Yes/No	Picking	Factual	Counting/Fluency
Yes/No	21313	0	1165	0
Picking	602	0	147	0
Factual	2246	0	62343	0
Counting/Fluency	1166	0	26198	0

Figure 4.7 Accuracy and confusion matrix tables for setup A



Accuracy

Class	Accuracy	# Samples
Yes/No	0.9282	22,478
Picking	0	749
Factual	0.857	64,589
Counting/Fluency	0.0223	27,364

Confusion Matrix

label \ prediction	Yes/No	Picking	Factual	Counting/Fluency
Yes/No	20863	0	1615	0
Picking	618	0	131	0
Factual	7430	0	55351	1808
Counting/Fluency	3811	0	22942	611

Accuracy

Class	Accuracy	# Samples
Yes/No	0.502	22,478
Picking	0	749
Factual	0.9072	64,589
Counting/Fluency	0.0462	27,364

Confusion Matrix

label \ prediction	Yes/No	Picking	Factual	Counting/Fluency
Yes/No	11283	0	11104	91
Picking	433	0	314	2
Factual	2008	0	58595	3986
Counting/Fluency	977	0	25123	1264

Accuracy

Class	Accuracy	# Samples
Yes/No	0.8117	22,478
Picking	0	749
Factual	0.8749	64,589
Counting/Fluency	0.0511	27,364

Confusion Matrix

label \ prediction	Yes/No	Picking	Factual	Counting/Fluency
Yes/No	18245	0	4189	44
Picking	543	0	206	0
Factual	3934	0	56508	4147
Counting/Fluency	2032	0	23935	1397

Figure 4.8 Accuracy and confusion matrix tables for setup B

Figures 4.5 and 4.6 illustrate the training graphs, displaying the values of loss, validation loss, accuracy, and validation accuracy for each training. Overall, Setup A demonstrates more consistent training progress compared to Setup B. Despite their differences in consistency, both setups ultimately converge towards similar results. To further evaluate the performance of the models, accuracy and confusion matrix tables for both setups are also provided in Figures 4.7 and 4.8. These tables are generated after the evaluation process, which is conducted immediately after the completion of training for each iteration. By analysing these tables, a comprehensive assessment of the classifiers' predictive capabilities and their ability to classify the classes in the QCoC dataset can be made.

Referring to Figures 4.7 and 4.8, accuracy scores for each of the four QCoC classes and the corresponding confusion matrix tables for each evaluation iteration are presented. Overall, Setup A demonstrates more consistency in accuracy scores and True Positive (TP) values for each class compared to Setup B. For both setups, the *Picking* class consistently achieves a zero accuracy score. This is expected since the total number of data points for the *Picking* class is only 0.65% (749 out of the overall 115,180 data points), making it challenging for the classifier to accurately learn patterns for this class. Among all classes, the *Factual* class consistently performs the best in both setups, with consistently high accuracy scores. The *Yes/No* class also exhibits good performance in Setup A, but its performance is less consistent in Setup B. On the other hand, the *Counting/Fluency* class shows poor performance in both setups, with zero accuracy scores in all iterations for Setup A, and only small accuracy scores (less than 10%) in all iterations for Setup B. This indicates that the classifier struggles to correctly classify instances from the *Counting/Fluency* class in both setups.

Looking at the overall performance in comparison to the baseline accuracy and F1 score, the classifier demonstrates a significant improvement against the baseline scores. To recap, the baseline scores for QCoC classification are 0.390 for accuracy and 0.561 for the F1 score. Tables 4.21 and 4.22 present the accuracy and F1 score for all iterations in Setup A and Setup B, respectively.

Table 4.21 Accuracy and F1 score for setup A

Metric	**Iteration 1**	**Iteration 2**	**Iteration 3**
Accuracy	0.571	0.571	0.570
F1	0.727	0.727	0.726
Average accuracy		**0.571**	
Average F1		**0.727**	

Table 4.22 Accuracy and F1 score for setup B

Metric	**Iteration 1**	**Iteration 2**	**Iteration 3**
Accuracy	0.498	0.443	0.489
F1	0.665	0.614	0.657
Average accuracy		**0.477**	
Average F1		**0.645**	

Referring to Table 4.21, Setup A has shown an increase of 0.181 (0.571 minus 0.390) or 18% in accuracy and 0.166 (0.727 minus 0.561) or 17% in F1 score, while Table 4.22 demonstrates that Setup B has an increase of 0.087 (0.477 minus 0.390) or 9% in accuracy and 0.084 (0.645 minus 0.561) or 8% in F1 score (calculated from average accuracy and F1).

From another perspective, Setup A has achieved a 46% increment from the baseline accuracy score (the percentage of 0.181/0.390) and a 30% increment from the baseline F1 score (the percentage of 0.166/0.561), whereas Setup B has achieved a 22% increment from the baseline accuracy score (the percentage of 0.087/0.390) and a 15% increment from the baseline F1 score (the percentage of 0.084/0.561). To visualize these information, Figure 4.9 presents a bar chart depicting the baseline accuracy and F1 scores, along with the accuracy and F1 scores for setups A and B. In the chart, the light green bars represent the baseline values, while the dark green bars represent the actual results for both experimental setups. Referring to this chart, it is evident that both setups significantly improved the evaluated scores, albeit with different parameter configurations. From an overall perspective, these results justify the proposed modified word vectors algorithm's success in incorporating external weights into the original word representation vector, with the objective of modifying its weightage to lean towards the intended contextual value.

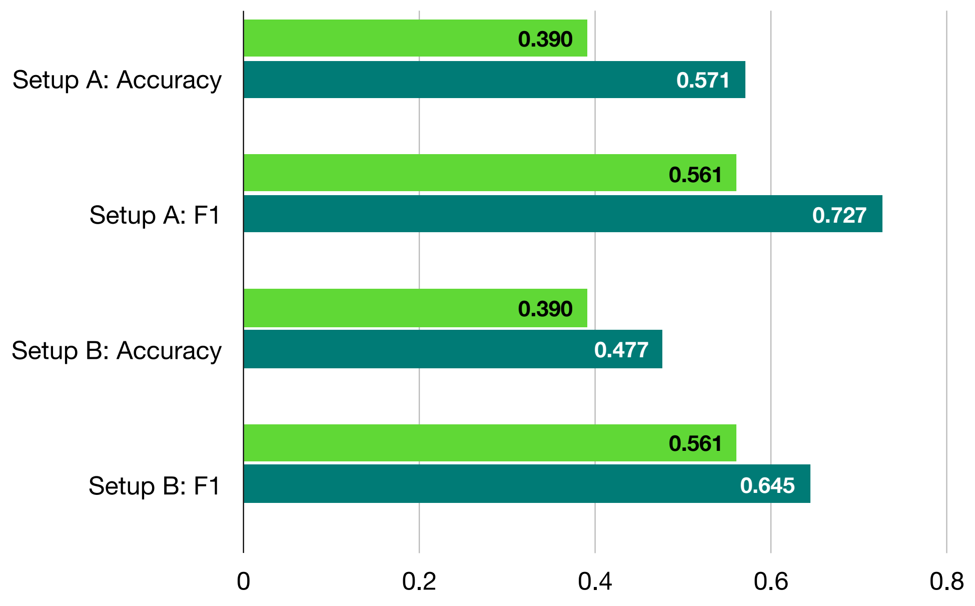


Figure 4.9 Baseline and result values for accuracy and F1 score

4.3 Discussion of Findings

The previous *Result Analysis* section has presented and discussed all results within the conducted five-phase research methodology. The overall findings derived from this result analysis has generally addressed the defined research objectives and offered valuable insights into the research hypothesis that *the word representation vector derived from the LLM can be altered using an external scalar weight, which can later be used as input for an ML model to perform text classification task.* To elaborate, the following are the discussion of findings from the presented results in relation to their contribution to achieving the three defined research objectives.

1. **Objective 1**: To develop a new algorithm that incorporate an external scalar weight into the word representation vector.

   The results from phase 3 of the research methodology have demonstrated the achievement of this objective through the proposed modified word vectors algorithm, Algorithm 6, as presented in Chapter 3. The results for the proposed algorithm have shown that by incorporating an external scalar weight value, the word representation vector can be semantically or contextually modified according to its importance. To recap, the research question posed for this objective is (RQ1): *How can an algorithm effectively incorporate external scalar weights into word representation vectors to enhance context understanding in a contextual text classification problem*? In proposing the algorithm to achieve this objective, two effectiveness merits are targeted: accuracy and adaptability. In terms of accuracy, the proposed algorithm should enhance the contextual value of the original word vector with respect to the targeted text classification problem, as mentioned in RQ1. Regarding adaptability, the proposed algorithm should maintain the original word vector's dimension, enabling it to be adaptable to different dimensions of the word vector. For both effectiveness merits, the results presented for Algorithm 6 have demonstrated its capability to satisfy both criteria, thus justifying the achievement of defined Objective 1 of this study. Zooming back into the broader aims, the proposed algorithm addresses the gap stated in the problem statement of this study, namely the inflexibility of externally modifying the weight of LLM-generated word vectors without modifying, re-training, or fine-tuning the LLM model. With the proposed algorithm, the computationally expensive LLM will only be used to generate the raw word vectors, while the modification of weights to suit the intended case study can be performed externally using a much less computationally expensive method.

2. **Objective 2:** To develop a new text classification dataset that emphasises differences in context representation.

   The attainment of this objective occurred during phase 2 of the research methodology through the development of the QCoC dataset. While not as prominent as the primary objective 1, this objective holds significance within the selected case study field, particularly in the realm of multiclass classification of question text within the Question-Answering (QA) and question classification domain, as extensively discussed in the Literature Review chapter of this thesis. The research question posed for this objective is (RQ2): *What criteria should be considered in the creation of a text classification dataset to emphasise differences in context representation*? As previously outlined, the chosen domain for the creation of the new text classification dataset is the QA system. Consequently, three benchmarked QA datasets were analysed in phase 1, and an algorithm was proposed in phase 2 to create the new dataset. In terms of the criteria outlined in

RQ2, the focal point in developing the QCoC dataset was the significance of both the question and general words within a question text, which are crucial factors in accentuating differences in the context representation of a question texts. To effectively represent this context, the QCoC dataset was created using Algorithm 1, which is defined as the algorithm for classifying the QCoC classes. The outcome of this algorithm is a question text classification dataset that underscores variations in how question text is classified based on its expected answer

3. **Objective 3**: To evaluate the developed algorithm using a Machine Learning (ML) model in the contextual text classification problem.

   The posed research question for this objective is (RQ3): How does the developed algorithm, incorporating external scalar weights, perform when applied to a contextual text classification task? In addressing this question, phase 5 of the research methodology evaluates the developed machine learning classifier using four evaluation metrics, namely Accuracy, Precision, Recall, and F1 score. In retrospect, all previously proposed methods in prior methodology phases are collectively assessed in this final phase. Two setups were evaluated, with one aiming for the highest accuracy performance, and another focusing on generalising the classifier's performance (increasing the precision score). Overall, when compared to the baseline values, the classifier’s results exhibit significant improvements, justifying the contribution of the previously developed modified word vectors algorithm in incorporating external scalar weight values to enhance the performance of the machine learning classifier. Looking back, this overall result suggests that the hypothesis “the word representation vector derived from the LLM can be altered using an external scalar weight, which can later be used as input for an ML model to perform text classification tasks”, can be validated with the proposed algorithm for Objective 1 of this study. With these findings, it can be concluded that the overall aims of this study to provide an alternative method to modify word representation vector weight, requiring fewer computational resources, have been achieved.

4.4 Summary

This chapter explores the results and findings obtained through the structured five-phase research methodology employed in this study. The chapter comprises two main sections: result analysis and discussion of findings. In the result analysis section, the results presented for each phase are discussed cohesively, as the outcomes of prior phases influence subsequent activities and processes in the following phases. Despite this interdependency, each phase's results are thoroughly analysed and presented to showcase all outcomes from the activities outlined in Chapter 3 (Methodology) of this thesis. To further contextualize the analysis in relation to the defined objectives of this study, the subsequent discussion of findings section provides broad perspectives into the implications of the results, aligning them with the general aims and hypothesis of the study. By elaborating on the findings for each objective, this section specifically addresses the contributions of each phase in the research methodology towards achieving the study's objectives, thereby providing clear indications of which methods or algorithms contributed to each objective. Overall, this chapter offers a focused discussion on the results and findings of this study, excluding external factors that influenced it. These external factors will be cumulatively addressed in the next and final chapter of this thesis as part of the conclusion to this study.

CHAPTER 5

CONCLUSION

5.1 Introduction

This chapter serves as the conclusion to the thesis, encompassing discussions on constraints and limitations, contributions, threats to validity, and avenues for future work. Firstly, constraints and limitations will be addressed to provide insight into the scope of this study. This acknowledgment is essential for understanding the boundaries within which the research was conducted. Following this, the chapter will outline the contributions made by the study, emphasising the areas where it has had the most significant impact. By highlighting these contributions, the chapter aims to underscore the value and relevance of the research findings. Subsequently, threats to validity will be discussed, focusing on factors that may compromise the study's results. Finally, the chapter will explore several potential paths for future research, suggesting areas where further investigation could expand upon the findings of this study.

5.2 Constraints and Limitations

As with any research, there are both constraints and limitations to acknowledge. One main constraint of this study is the narrow focus of the experiments conducted, which centered on a specific contextual text data classification task which is the question type classification using the QCoC dataset. While the proposed method exhibited promising results within this scope, its applicability to other types of text data classification tasks warrants further exploration and validation. Moreover, although the size and diversity of the QCoC dataset sufficed for the current study, it may not fully represent the breadth of question types and contexts encountered in real-world applications, necessitating the assessment of the proposed method's generalizability with more extensive and diverse datasets.

Next, it's essential to recognise that the experiments were conducted using a consumer-grade computer with specific hardware and software configurations. Therefore, the performance and efficiency of the proposed method may vary across different computing systems, particularly when dealing with larger datasets or more complex models. Thus, considerations regarding the scalability and generalizability of the proposed method become imperative when applying it to diverse settings and configurations. Lastly, while the proposed modified word vector method demonstrated success in this study, its effectiveness might be influenced by very specific or domain-specific vocabularies. Variations in corpus and language use could impact its efficacy, necessitating further investigation to identify potential limitations and avenues for improvement.

5.3 Contributions

Overall, this study makes several significant contributions to the field of contextual text data classification within the NLP research domain:

1. **Proposed modified word vectors algorithm**: The development of the modified word vectors algorithm stands out as the main contribution of this study. This algorithm enables the modification of LLM-generated word representation vectors externally, eliminating the need to modify, retrain, or fine-tune the original pretrained LLM. This contribution is particularly valuable in contexts where computing resources are scarce, such as smaller research teams or educational institutions with limited access to high-performance computing resources. By showcasing the effectiveness of this algorithm, it becomes evident that word representation vectors can be modified externally. This facilitates machine learning classification algorithms with fewer parameters, significantly reducing the demand for extensive computing resources during training.

2. **The QCoC dataset**: This study leverages the benchmarked QA dataset named CoQA to develop a new multiclass text classification dataset known as QCoC (Question Classification of CoQA). Unlike existing taxonomy-based datasets, QCoC is specifically designed to address a gap identified in the literature surrounding Question Type Classification (QTC) datasets. Specifically, QCoC focuses on mitigating the limitation of current QTC datasets, which overlook the

phenomenon of abstractive answers in QA datasets. This unique feature of QCoC aims to enhance the relevance and applicability of the dataset, particularly in contexts where more nuanced and abstract responses are common. By filling this gap, QCoC contributes to advancing the field of question classification by providing a dataset that better reflects the complexities and nuances of real-world question-answering scenarios.

3. **Efficient question-type classifier**: The research successfully developed an efficient question type classifier using a vanilla feed-forward neural network (NN). The classifier achieved significantly higher accuracy and F1 scores compared to baseline scores, while maintaining a small number of trainable parameters. This economical question-type classifier has the potential to be deployed in various applications where computational resources are limited or costly, without compromising performance. However, it is important to acknowledge that such a classifier will not work solely on LLM-generated word embeddings/vectors as input. This is showcased in the results of *Activity 2 within Phase 3* of the research methodology, where the original LLM-generated embeddings demonstrated a very minuscule scale of vector differences between one sentence and another. In other words, without an external weight and a method to incorporate that weight into the vector embeddings, such a classifier may not yield good results if based solely on the original LLM-generated word representation vectors/embeddings.

4. **Transferability to other domains**: The proposed modified word vectors method demonstrated high transferability to other domains of contextual text data classification beyond question type classification. This is primarily attributed to the external component of this method, which is the scalar weight values. As demonstrated in *Activity 1 within Phase 3* of the research methodology, these scalar weight values can be derived from a simple mathematical calculation, such as the term frequency percentage. Therefore, theoretically, utilising the same modified word vectors method with different scalar weights will yield similar results. Moreover, the method's capability to maintain fixed-length vector embeddings despite varying sentence lengths enhances its versatility in handling diverse NLP tasks.

5. **Practical implications**: The research findings have practical implications for various NLP applications, particularly in scenarios where computational resources are limited or costly. By offering a cost-effective alternative to fine-tuning LLMs, the proposed methods opens up opportunities for efficient and economical text classification solutions across other NLP domains such as sentiment analysis, topic categorization, and intent recognition.

5.4 Threats to Validity

The validity of any research is crucial to ensure the credibility and reliability of the findings. In this study, several threats to validity need to be addressed to ensure the robustness of the proposed methods and algorithms for the contextual text classification process.

1. **Internal Validity**: One potential threat to internal validity is related to the experimental setup and the choice of hyperparameters during the experiments. To mitigate this, the experiments were conducted multiple times, and the results were analysed for consistency. Additionally, the random initialization of the neural network weights was performed to minimize bias.

2. **External Validity**: Generalization of the findings to other datasets and domains could be a potential external validity threat. Although the proposed methods and algorithms demonstrated promising results on the QCoC dataset, its performance on different datasets needs to be explored to establish its broader applicability.

3. **Construct Validity**: Ensuring that the proposed modified word vectors method accurately reflects the semantic significance of words is essential. To address this concern, the method was first evaluated on a sample case study before being applied to the QCoC dataset. However, it is important to note that both cases involved question text data. Therefore, for application in other contexts, further investigation is necessary to validate the method's effectiveness and generalizability.

4. **Conclusion Validity**: The sample size for the conducted experiments in this study may be limited or insufficient in certain cases, and thus, the conclusions derived from these experiments should be interpreted with caution. Conducting

experiments on larger datasets and using more diverse question types could enhance the generalizability of the findings.

5. **Reproducibility**: To ensure the reproducibility of the results, all experimental setups, hyperparameters, and model architectures have been thoroughly documented. The pseudocode and datasets used in the research will be made available for public access.

Despite these potential threats, the research design, thorough experimentation, and detailed analysis contribute to the robustness of the proposed methodology. Additionally, future studies should explore further validation of different datasets and domains to strengthen the overall validity of the research findings.

5.5 Future Works

Though this research has showcased a notable contribution in the field of NLP, there are several potential avenues for future work and improvement. This section outlines potential directions for further research and development, building upon the foundations laid by this study.

1. **Fine-tuning on larger datasets**: Experimenting with fine-tuning the proposed methods on larger datasets could provide deeper insights into its scalability and robustness. Evaluating the classifier's performance on diverse and more extensive datasets will help assess its generalization capabilities and potential for wider applications.

2. **Comparison with state-of-the-art ML models**: Conducting a thorough comparison between the proposed methods and state-of-the-art ML models on various benchmark datasets will offer a comprehensive understanding of its competitive advantage. This evaluation will allow researchers and practitioners to identify the contexts in which the proposed methodology excels and areas that might need further improvement.

3. **Exploration of other weighting methods**: Investigating alternative word vector weighting methods could enhance the performance and flexibility of the proposed methodology. Exploring different weighting techniques and experimenting with

various scalar weight values may yield more optimal results for specific NLP downstream tasks.

4. **Application to domain-specific classification**: Applying the developed methods to domain-specific text classification tasks, such as medical or financial text data, could demonstrate its adaptability and efficacy in specialised contexts. These domain-specific applications will help uncover the methodology's strengths and limitations in real-world scenarios.

5. **Deployment in practical applications**: Deploying the developed question-type classifier in practical applications, such as chatbots or virtual assistants, will offer insights into its real-world usability and impact of the developed methods. Evaluating its performance in real-time scenarios will be crucial for assessing its practicality and user experience.

5.6 Summary

This chapter serves as the concluding remarks for this thesis, encompassing discussions on constraints and limitations, contributions, threats to validity, and future works. Building upon the proposed methods and algorithms, along with the thoroughly discussed results and findings in prior sections, it addresses constraints and limitations that should be acknowledged for future research within the field. Despite its limitations, this study contributes valuable knowledge to the field, particularly in word representation vector modification and contextual text classification. Through thorough discussion, these contributions are poised to advance the field and pave the way for new research paths. Additionally, various threats to validity are examined, including internal, external, construct, conclusion, and reproducibility concerns. Lastly, avenues for future works are discussed to suggest potential paths for further exploration and investigation, with the hope of contributing to the identification of potential gaps within the field.

REFERENCES

Abubakar, H. D., Umar, M., & Bakale, M. A. (2022). Sentiment classification: Review of text vectorization methods: Bag of words, Tf-Idf, Word2vec and Doc2vec. *SLU Journal of Science and Technology*, *4*(1 & 2), 27-33. doi:10.56471/slujst.v4i.266

Alammary, A. S. (2022). BERT models for Arabic text classification: a systematic review. *Applied Sciences*, *12*(11), 5720. doi:10.3390/app12115720

Ameer, I., Bölücü, N., Siddiqui, M. H. F., Can, B., Sidorov, G., & Gelbukh, A. (2023). Multi-label emotion classification in texts using transfer learning. *Expert Systems with Applications*, *213*, 118534. doi:10.1016/j.eswa.2022.118534

Apidianaki, M. (2023). From word types to tokens and back: A survey of approaches to word meaning representation and interpretation. *Computational Linguistics*, *49*(2), 465-523.

Arora, S., Hu, W., & Kothari, P. K. (2018, July). An analysis of the t-sne algorithm for data visualization. In *Conference on learning theory* (pp. 1455-1462). PMLR.

Asudani, D. S., Nagwani, N. K., & Singh, P. (2023). Impact of word embedding models on text analytics in deep learning environment: a review. *Artificial intelligence review*, *56*(9), 10345-10425. doi:10.1007/s10462-023-10419-1

Azunre, P. (2021). *Transfer Learning for Natural Language Processing*. Manning.

Badri, N., Kboubi, F., & Chaïbi, A. H. (2022). Combining FastText and Glove Word Embedding for Offensive and Hate speech Text Detection. *Procedia Computer Science*, *207*, 769–778. doi:10.1016/j.procs.2022.09.132

Bharadiya, J. P. (2023). Transfer Learning in Natural Language Processing (NLP). *European Journal of Technology*, *7*(2), 26–35. doi:10.47672/ejt.1490

Bilal, M., & Almazroi, A. A. (2022). Effectiveness of fine-tuned BERT model in classification of helpful and unhelpful online customer reviews. *Electronic Commerce Research*, *23*(4), 2737–2757. doi:10.1007/s10660-022-09560-w

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, *3*, 993–1022. doi:10.5555/944919.944937

Bojanowski, P., Grave, É., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146. doi:10.1162/tacl_a_00051

Bollegala, D., & O'Neill, J. (2022, April 25). *A survey on Word Meta-Embedding Learning*. arXiv.org. https://arxiv.org/abs/2204.11660

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., … Amodei, D. (2020). Language Models are Few-Shot Learners. In H. Larochelle, M.

Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 1877–1901). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., St John, R., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y., Strope, B., & Kurzweil, R. (2018, March 29). *Universal Sentence Encoder*. arXiv.org. https://arxiv.org/abs/1803.11175

Chen, Q., Zhang, R., Zheng, Y., & Mao, Y. (2022, January 21). *Dual Contrastive Learning: text classification via Label-Aware data augmentation*. arXiv.org. http://arxiv.org/abs/2201.08702

Choi, E., He, H., Iyyer, M., Yatskar, M., Yih, W., Choi, Y., Liang, P., & Zettlemoyer, L. (2018, August 21). *QUAC : Question answering in context*. arXiv.org. https://arxiv.org/abs/1808.07036

Chotirat, S., & Meesad, P. (2021). Part-of-Speech tagging enhancement to natural language processing for Thai wh-question classification with deep learning. *Heliyon*, *7*(10), e08216. doi:10.1016/j.heliyon.2021.e08216

Church, K. (2016). Word2Vec. *Natural Language Engineering*, *23*(1), 155–162. doi:10.1017/s1351324916000334

Church, K., Chen, Z., & Ma, Y. (2021). Emerging trends: A gentle introduction to fine-tuning. *Natural Language Engineering*, *27*(6), 763–778. doi:10.1017/s1351324921000322

Da Costa, L. S., Oliveira, I. L., & Fileto, R. (2023). Text classification using embeddings: a survey. *Knowledge and Information Systems*, *65*(7), 2761–2803. doi:10.1007/s10115-023-01856-z

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018, October 11). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv.org. https://arxiv.org/abs/1810.04805

Dharma, E. M., Gaol, F. L., Warnars, H. L. H. S., & Soewito, B. (2022). The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. *Journal of Theoretical and Applied Information Technology*, 100(2), 31.

Dogra, V., Verma, S., Kavita, K., Chatterjee, P., Shafi, J., Choi, J., & Ijaz, M. F. (2022). A complete process of text classification system using State-of-the-Art NLP models. *Computational Intelligence and Neuroscience*, *2022*, 1–26. doi:10.1155/2022/1883698

Duc, T. L., Leiva, R. G., Casari, P., & Östberg, P.-O. (2019). Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey. *ACM Comput. Surv.*, *52*(5). doi:10.1145/3341145

Elnagar, A., Yagi, S., Mansour, Y., Lulu, L., & Fareh, S. (2023). A benchmark for evaluating Arabic contextualized word embedding models. *Information Processing & Management*, *60*(5), 103452. doi:10.1016/j.ipm.2023.103452

Evangelopoulos, N. (2013). Latent semantic analysis. *Wiley Interdisciplinary Reviews. Cognitive Science*, *4*(6), 683–692. doi:10.1002/wcs.1254

Fernandez, R. C., Elmore, A. J., Franklin, M. J., Krishnan, S., & Tan, C. (2023). How large language models will disrupt data management. *Proceedings of the VLDB Endowment*, *16*(11), 3302–3309. doi:10.14778/3611479.3611527

Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis (Special Volume of the Philological Society)*, *1952*(59), 1–32.

Fodor, J., De Deyne, S., & Suzuki, S. (2023, June). The Importance of Context in the Evaluation of Word Embeddings: The Effects of Antonymy and Polysemy. *Proceedings of the 15th International Conference on Computational Semantics* (pp. 155-172).

Gasparetto, A., Marcuzzo, M., Zangari, A., & Albarelli, A. (2022). A survey on Text Classification Algorithms: From Text to Predictions. *Information*, *13*(2), 83. doi:10.3390/info13020083

Ghosal, S., & Jain, A. (2022). Weighted aspect based sentiment analysis using extended OWA operators and Word2Vec for tourism. *Multimedia Tools and Applications*, *82*(12), 18353–18380. doi:10.1007/s11042-022-13800-4

Golzari, S., Sanei, F., Saybani, M. R., Harifi, A., & Basir, M. A. (2021). Question Classification in Question Answering System using Combination of Ensemble Classification and Feature Selection. *Journal of AI and Data Mining*. doi:10.22044/jadm.2021.10016.2142

Gupta, D., Pujari, R., Ekbal, A., Bhattacharyya, P., Maitra, A., Jain, T., & Sengupta, S. (2021, January 20). *Can Taxonomy Help? Improving Semantic Question Matching using Question Taxonomy*. arXiv.org. https://arxiv.org/abs/2101.08201

Hamza, A., En-Nahnahi, N., Zidani, K. A., & Ouatik, S. E. A. (2021). An arabic question classification method based on new taxonomy and continuous distributed representation of words. *Journal of King Saud University-Computer and Information Sciences*, *33*(2), 218-224. doi:10.1016/j.jksuci.2019.01.001

Harris, Z. S. (1954). Distributional Structure. *Word*, *10*(2–3), 146–162. doi:10.1080/00437956.1954.11659520

Hossain, M. R., Hoque, M. M., & Siddique, N. (2023). Leveraging the meta-embedding for text classification in a resource-constrained language. *Engineering Applications of Artificial Intelligence*, *124*, 106586. doi:10.1016/j.engappai.2023.106586

Incitti, F., Urli, F., & Snidaro, L. (2023). Beyond word embeddings: A survey. *Information Fusion*, *89*, 418–436. doi:10.1016/j.inffus.2022.08.024

Jalilifard, A., Caridá, V. F., Mansano, A. F., Cristo, R. S., & da Fonseca, F. P. C. (2021). Semantic Sensitive TF-IDF to Determine Word Relevance in Documents. *Advances in Computing and Network Communications*, 327–337. doi:10.1007/978-981-33-6987-0_27

Jardim, R., Delgado, C., & Schneider, D. (2022). Data science supporting a question classifier model. *Procedia Computer Science*, *199*, 1237–1243. doi:10.1016/j.procs.2022.01.157

Jiao, Q., & Zhang, S. (2021). A Brief Survey of Word Embedding and Its Recent Development. *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, *5*, 1697–1701. doi:10.1109/IAEAC50856.2021.9390956

Johnson, S. J., Murty, M. R., & Navakanth, I. (2024). A detailed review on word embedding techniques with emphasis on word2vec. *Multimedia Tools and Applications*, *83*(13), 37979–38007. doi:10.1007/s11042-023-17007-z

Jones, K. S. (2004). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation, 60*(5), 493-502. doi:10.1108/00220410410560573

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). *Fasttext. zip: Compressing text classification models*. arXiv.org. https://arxiv.org/abs/1612.03651

Ju, Y., Zhao, F., Chen, S., Zheng, B., Yang, X., & Liu, Y. (2019). Technical report on conversational question answering. arXiv.org. https://arxiv.org/abs/1909.10772

Kanakarajan, K. R., Kundumani, B., & Sankarasubbu, M. (2021). *Small-Bench NLP: Benchmark for small single GPU trained models in Natural Language Processing*. arXiv.org. https://arxiv.org/abs/2109.10847

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *35*(3), 400–401. doi:10.1109/TASSP.1987.1165125

Khurana, D., Koli, A., Khatter, K., & Singh, S. (2023). Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, *82*(3), 3713–3744. doi:10.1007/s11042-022-13428-4

Kici, D., Malik, G., Cevik, M., Parikh, D., & Basar, A. (2021, June). A BERT-based transfer learning approach to text classification on software requirements specifications. In *Canadian Conference on AI* (Vol. 1, p. 04207).

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text Classification Algorithms: A Survey. *Information, 10*(4). doi:10.3390/info10040150

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). *Albert: A lite bert for self-supervised learning of language representations*. arXiv.org. https://arxiv.org/abs/1909.11942.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. doi:10.1038/nature14539

Le, Q., & Mikolov, T. (2014, June). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, (pp. 1188-1196). Retrieved from https://proceedings.mlr.press/v32/le14.html

Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., ... & He, L. (2022a). A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST), 13*(2), 1-41. doi:10.1145/3495162

Li, X., & Roth, D. (2002b). Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*. https://aclanthology.org/C02-1150.pdf

Lippmann, R. (1994). Book Review: 'Neural Networks, A Comprehensive Foundation', by Simon Haykin. *International Journal of Neural Systems, 05*(04), 363–364. doi:10.1142/S0129065794000372

Liu, S., Zhang, X., Zhang, S., Wang, H., & Zhang, W. (2019). Neural Machine Reading Comprehension: Methods and Trends. *Applied Sciences*, *9*(18), 3698. doi:10.3390/app9183698

Liu, W., Pang, J., Li, N., Yue, F., & Liu, G. (2023). Few-shot short-text classification with language representations and centroid similarity. *Applied Intelligence, 53*(7), 8061-8072.

Liu, X., Sun, T., He, J., Wu, J., Wu, L., Zhang, X., Jiang, H., Cao, Z., Huang, X., & Qiu, X. (2022). Towards efficient NLP: A standard evaluation and A strong baseline. *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 3288–3303. https://arxiv.org/abs/2110.07038

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., & Allen, P. G. (2019). *Roberta: A robustly optimized bert pretraining approach*. arXiv.org. https://arxiv.org/abs/1907.11692

Liu, Z., Lin, Y., & Sun, M. (2023). *Representation Learning for Natural Language Processing* (p. 521). Springer Nature.

Lund, K., & Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, *28*(2), 203–208. doi:10.3758/BF03204766

Mars, M. (2022). From Word Embeddings to Pre-Trained Language Models: A State-of-the-Art Walkthrough. *Applied Sciences*, *12*(17). doi:10.3390/app12178805

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space.* arXiv.org. https://arxiv.org/abs/1301.3781

Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., … Roth, D. (2023). Recent Advances in Natural Language Processing via Large Pre-trained Language Models: A Survey. *ACM Comput. Surv., 56*(2). doi:10.1145/3605943

Mundotiya, R. K., Mehta, A., Baruah, R., & Singh, A. K. (2022). Integration of morphological features and contextual weightage using monotonic chunk attention for part of speech tagging. *Journal of King Saud University - Computer and Information Sciences, 34*(9), 7324–7334. doi:10.1016/j.jksuci.2021.08.023

Nandanwar, A. K., & Choudhary, J. (2023). Contextual Embeddings-Based Web Page Categorization Using the Fine-Tune BERT Model. *Symmetry*, *15*(2). doi:10.3390/sym15020395

Naseem, U., Razzak, I., Khan, S. K., & Prasad, M. (2021). A Comprehensive Survey on Word Representation Models: From Classical to State-of-the-Art Word Representation Language Models. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, *20*(5). doi:10.1145/3434237

Onita, D. (2023). Active Learning Based on Transfer Learning Techniques for Text Classification. *IEEE Access*, *11*, 28751–28761. doi:10.1109/ACCESS.2023.3260771

Patil, R., Boit, S., Gudivada, V., & Nandigam, J. (2023). A Survey of Text Representation and Embedding Techniques in NLP. *IEEE Access*, *11*, 36120–36146. doi:10.1109/ACCESS.2023.3266377

Pennington, J., Socher, R., & Manning, C. (2014, October). GloVe: Global Vectors for Word Representation. In A. Moschitti, B. Pang, & W. Daelemans (Eds.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). doi:10.3115/v1/D14-1162

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018, June). Deep Contextualized Word Representations. In M. Walker, H. Ji, & A. Stent (Eds.), *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (pp. 2227–2237). doi:10.18653/v1/N18-1202

Pham, P., Nguyen, L. T., Pedrycz, W., & Vo, B. (2023). Deep learning, graph-based text representation and classification: a survey, perspectives and challenges. *Artificial Intelligence Review, 56*(6), 4893-4927. doi: 10.1007/s10462-022-10265-7

Qasim, R., Bangyal, W. H., Alqarni, M. A., & Ali Almazroi, A. (2022). A Fine-Tuned BERT-Based Transfer Learning Approach for Text Classification. *Journal of Healthcare Engineering*, *2022*(1), 3498123. doi:10.1155/2022/3498123

Qin, C., Zhang, A., Zhang, Z., Chen, J., Yasunaga, M., & Yang, D. (2023). *Is ChatGPT a general-purpose natural language processing task solver?* arXiv.org. https://arxiv.org/abs/2302.06476

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. *Preprint.* 1-12. Retrieved from https://www.mikecaptain.com/resources/pdf/GPT-1.pdf

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8). Retrieved from https://openai.com/blog/better-language-models/

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., … Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research, 21*(140), 1–67. Retrieved from http://jmlr.org/papers/v21/20-074.html

Rajpurkar, P., Jia, R., & Liang, P. (2018, July). Know What You Don't Know: Unanswerable Questions for SQuAD. In I. Gurevych & Y. Miyao (Eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 784–789). doi:10.18653/v1/P18-2124

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016, November). SQuAD: 100,000+ Questions for Machine Comprehension of Text. In J. Su, K. Duh, & X. Carreras (Eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (pp. 2383–2392). doi:10.18653/v1/D16-1264

Reddy, S., Chen, D., & Manning, C. D. (2019). CoQA: A Conversational Question Answering Challenge. *Transactions of the Association for Computational Linguistics, 7*, 249–266. doi:10.1162/tacl_a_00266

Rosset, C. (2020). *Turing-NLG: A 17-billion-parameter language model by Microsoft - Microsoft Research*. Retrieved from https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

Sabri, T., Beggar, O. E., & Kissi, M. (2022). Comparative study of Arabic text classification using feature vectorization methods. *Procedia Computer Science, 198*, 269–275. doi:10.1016/j.procs.2021.12.239

Salim, M. N., & Mustafa, B. S. (2022, November). A survey on word representation in natural language processing. In *AIP Conference Proceedings* (Vol. 2394, No. 1). AIP Publishing.

Salton, G., & Lesk, M. E. (1968). Computer Evaluation of Indexing and Text Processing. *Journal of the ACM (JACM), 15*(1), 8–36. doi:10.1145/321439.321441

Sangodiah, A., Fui, Y. T., Heng, L. E., Jalil, N. A., Ayyasamy, R. K., & Meian, K. H. (2021). A Comparative Analysis On Term Weighting In Exam Question Classification. *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 199–206. doi:10.1109/ISMSIT52890.2021.9604639

Sezerer, E., & Tekir, S. (2021). *A Survey On Neural Word Embeddings*. arXiv.org. https://arxiv.org/abs/2110.01804

Sharir, O., Peleg, B., & Shoham, Y. (2020). *The cost of training nlp models: A concise overview*. arXiv.org. https://arxiv.org/abs/2004.08900

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). *Megatron-lm: Training multi-billion parameter language models using model parallelism*. arXiv.org. https://arxiv.org/abs/1909.08053

Singh, S., & Mahmood, A. (2021). The NLP Cookbook: Modern Recipes for Transformer Based Deep Learning Architectures. *IEEE Access, 9*, 68675–68702. doi:10.1109/ACCESS.2021.3077350

Soni, S., Chouhan, S. S., & Rathore, S. S. (2023). TextConvoNet: a convolutional neural network based architecture for text classification. *Applied Intelligence, 53*(11), 14249–14268. doi:10.1007/s10489-022-04221-9

Storks, S., Gao, Q., & Chai, J. Y. (2019). *Recent Advances in Natural Language Inference: A Survey of Benchmarks, Resources, and Approaches*. arXiv.org. https://arxiv.org/abs/1904.01172

Umer, M., Imtiaz, Z., Ahmad, M., Nappi, M., Medaglia, C., Choi, G. S., & Mehmood, A. (2023). Impact of convolutional neural network and FastText embedding on text classification. *Multimedia Tools and Applications, 82*(4), 5569–5585. doi:10.1007/s11042-022-13459-x

Uszkoreit, J. (2017). *Transformer: A Novel Neural Network Architecture for Language Understanding*. Retrieved from https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Uymaz, H. A., & Metin, S. K. (2022). Vector based sentiment and emotion analysis from text: A survey. *Engineering Applications of Artificial Intelligence, 113*, 104922. doi:10.1016/j.engappai.2022.104922

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S.

Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 30). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wadud, M. A. H., Mridha, M. F., & Rahman, M. M. (2022). Word embedding methods for word representation in deep learning for natural language processing. *Iraqi Journal of Science, 63*(3), 1349-1361. doi: 10.24996/ijs.2022.63.3.37

Wadud, M.A.H., Mridha, M.F., Shin, J., Nur, K., Saha, A.K. (2023). Deep-BERT: Transfer Learning for Classifying Multilingual Offensive Texts on Social Media. *Computer Systems Science and Engineering, 44*(2), 1775-1791. doi: 10.32604/csse.2023.027841

Wahba, Y., Madhavji, N., & Steinbacher, J. (2023). A Comparison of SVM Against Pre-trained Language Models (PLMs) for Text Classification Tasks. Machine Learning, Optimization, and Data Science: *8th International Conference, LOD 2022, Certosa Di Pontignano, Italy, September 18–22, 2022, Revised Selected Papers, Part II*, 304–313. Presented at the Certosa di Pontignano, Italy. doi:10.1007/978-3-031-25891-6_23

Weaver, W. (1955). Translation. *Machine Translation of Languages*, *14*, 15–23. Retrieved from https://aclanthology.org/1952.earlymt-1.1.pdf

Weerakoon, C., & Ranathunga, S. (2021). Question Classification for the Travel Domain using Deep Contextualized Word Embedding Models. *2021 Moratuwa Engineering Research Conference (MERCon)*, 573–578. doi:10.1109/MERCon52712.2021.9525789

Worth, P. J. (2023). Word embeddings and semantic spaces in natural language processing. *International Journal of Intelligence Science, 13*(1), 1–21. doi: 10.4236/ijis.2023.131001

Xu, J., Xie, J., Cai, Y., Lin, Z., Leung, H.-F., Li, Q., & Chua, T.-S. (2024). Context-Aware Dynamic Word Embeddings for Aspect Term Extraction. *IEEE Transactions on Affective Computing, 15*(1), 144–156. doi:10.1109/TAFFC.2023.3262941

Yamada, I., Asai, A., Shindo, H., Takeda, H., & Matsumoto, Y. (2020). *LUKE: deep contextualized entity representations with entity-aware self-attention*. arXiv.org. https://arxiv.org/abs/2010.01057

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 32). Retrieved from https://proceedings.neurips.cc/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf

Yatskar, M. (2019, June). A Qualitative Comparison of CoQA, SQuAD 2.0 and QuAC. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 2318–2323). doi:10.18653/v1/N19-1241

Zhang, Y., Wang, M., Ren, C., Li, Q., Tiwari, P., Wang, B., & Qin, J. (2024, February). *Pushing the limit of LLM capacity for text classification*. arXiv.org. https://arxiv.org/abs/2402.07470

Zhang, Z., Yang, J., & Zhao, H. (2021). Retrospective Reader for Machine Reading Comprehension. *Proceedings of the AAAI Conference on Artificial Intelligence, 35*(16), 14506–14514. doi:10.1609/aaai.v35i16.17705

Zhao, J., Bao, J., Wang, Y., Zhou, Y., Wu, Y., He, X., & Zhou, B. (2021, November). RoR: Read-over-Read for Long Document Machine Reading Comprehension. In M.-F. Moens, X. Huang, L. Specia, & S. W.-T. Yih (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021* (pp. 1862–1872). doi:10.18653/v1/2021.findings-emnlp.160

Zhong, R., Snell, C., Klein, D., & Steinhardt, J. (2022). *Summarizing Differences between Text Distributions with Natural Language*. arXiv.org. https://arxiv.org/abs/2201.12323

Zhou, H. (2022a). Research of Text Classification Based on TF-IDF and CNN-LSTM. *Journal of Physics: Conference Series, 2171*(1), 012021. doi:10.1088/1742-6596/2171/1/012021

Zhou, M., Liu, D., Zheng, Y., Zhu, Q., & Guo, P. (2022b). A text sentiment classification model using double word embedding methods. *Multimedia Tools and Applications, 81*(14), 18993–19012. doi:10.1007/s11042-020-09846-x