School of Pure and Applied Sciences
Department of Computing and Informatics,

ADVANCING BERT PRUNING TECHNIQUES: A CONCEPTUAL FRAMEWORK AND AN ENSEMBLE METHOD OF NEURAL SUBNETWORKS.

by

NYALALANI SMARTS

Student ID number: 14100105

BA (Multimedia) (RMIT university), MSc (Information Systems) (BIUST)

A Dissertation/Thesis Submitted to the Faculty of Sciences in Partial Fulfilment of the Requirements for the Award of the Degree of Doctor of Philosophy in Computer Science of BIUST

Supervisor(s): Prof. Rajalakshmi Selvaraj,
Department of Computing and Informatics
School of pure and applied Sciences, BIUST
selvarajr@biust.ac.bw,
Supervisor(s): Prof. Venumadhav Kuthadi
Department of Computing and Informatics
School of pure and applied Sciences, BIUST
kuthadiv@biust.ac.bw

July 2025

BOTSWANA INTERNATIONAL UNIVERSITY OF SCIENCE & TECHNOLOGY

POSTGRADUATE SCHOOL

Declaration Regarding the Work and Copyright

DECLARATION REGARDING THE WORK AND COPYRIGHT

Candidate: NYALALANI SMARTS

Student ID: 14100105

Thesis Titled: **ADVANCING BERT PRUNING TECHNIQUES: A CONCEPTUAL FRAMEWORK AND AN ENSEMBLE METHOD OF NEURAL SUBNETWORKS**

I, **Nyalalani Smarts**, certify that the Thesis is all my own original work and that I have not obtained a degree in this University or elsewhere on the basis of any of this work.

(If the thesis is based on a group project, then the student must indicate the extent of her / his contribution, with reference to any other theses submitted or published by each collaborator in the project, and a declaration to this effect must be included in the thesis)

This dissertation/thesis is copyright material protected under the Berne Convention, the Copyright and Neighbouring Rights Act, Act. No. 8 of 2000 and other international and national enactments, in that behalf, on intellectual property. It must not be reproduced by any means, in full or in part, except for short extracts in fair dealing; for researcher private study, critical scholarly review or discourse with an acknowledgement, without the written permission of the office of the Postgraduate School, on behalf of both the author and the BIUST.

Signed: ---------- Date: **01 July 2025**

Primary Supervisor (please write in caps or type)
PROF RAJALAKSHMI SELVARAJ_____

I, the Candidate's **Primary Supervisor**, hereby confirm that I have inspected the above titled thesis and, to the best of my knowledge, it is based on the original work of the candidate.

Signed: ---------- Date: -----01/07/2025----------

BOTSWANA INTERNATIONAL UNIVERSITY OF SCIENCE & TECHNOLOGY

POSTGRADUATE SCHOOL

Antiplagiarism check

FORM PGS - 330

MEMORANDUM

Postgraduate candidate: NYALALANI SMARTS **Student ID:**14100105
Degree level: PhD [✓], or Master [ ] *(tick the appropriate)*

Thesis Titled: **ADVANCING BERT PRUNING TECHNIQUES: A CONCEPTUAL FRAMEWORK AND AN ENSEMBLE METHOD OF NEURAL SUBNETWORKS.**

The Antiplagiarism Check had been completed and the results is detailed below.

Software and version used: **TURNITIN FEEDBACK STUDIO**

Outcome: **8 % SIMILARITY INDEX SCORE**

Resolution: Comment from the Supervisor needed: YES [ ], or NOT [√] *(tick the appropriate)*

Officer who completed the check (please write in caps or type) **KELEBONYE BAGAI**

Signed: Date: **25/02/2025**

COMMENTS BY THE PRIMARY SUPERVISOR (*if requested*):

_____
_____

Primary supervisor (please write in caps or type) __PROF RAJALAKSHMI SELVARAJ__

Signed: ---------- Date: ---25/02/2025----------

CERTIFICATION

This document signatory endorses that he or she has reviewed the document and firmly recommends for approval by the Faculty of Sciences a thesis titled: **ADVANCING BERT PRUNING TECHNIQUES: A CONCEPTUAL FRAMEWORK AND AN ENSEMBLE METHOD OF NEURAL SUBNETWORKS**, in partial fulfillment of the degree of Doctor of Philosophy in Computer Science of the Botswana International University of Science and Technology (BIUST).

Supervisor: Prof. Rajalakshmi Selvaraj

Signature: _____

Co-Supervisor: Prof. Venumadhav Kuthadi

Signature: _____

ACKNOWLEDGEMENTS

Firstly, I would like to thank God for seeing me through my studies and for protecting my health all through. I am indebted to BIUST for agreeing to provide 50% tuition fee waiver for the full entire programme of Ph.D. I am truly grateful to my supervisors Prof Rajalakshmi Selvaraj and Prof Venumadhav Kuthadi for carving out time off schedule to avail themselves of their expertise, motivation, and encouragement. This research would be nothing if it wasn't their support and guidance. To my wife, I would like to express an extra deep gratitude for being exceptional throughout the journey of this PhD. We not only provided financial support to me but also, she stood by me emotionally and mentally. She was my confidence as I shared the progress, setbacks, and successes of this endeavor. Truly, she has been the most pivotal pillar of my strength. To my fellow postgraduates and friends at BIUST, I will be grateful for the encouragement you have given me. Your faith has inspired me to continue when this course as it was so rigorous I wanted to quit. And finally, to all those individuals who actively participated in this endeavor, all the good words, encouragement and help in the technical process, a HUGE thank you. This thesis represents the culmination of collective efforts, and I am overwhelmed with gratitude for the essential contributions each of you has made to this achievement. To all of you who stand by my side and who have been so incredibly generous, I say thank you.

DEDICATION

To My beautiful wife Ratanang and two daughters Thokgamo and Buthu.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS 5

DEDICATION 6

LIST OF TABLES 11

LIST OF FIGURES 12

LIST OF APPENDICES 13

ABSTRACT 14

Chapter 1: Introduction and Motivation 16

1.5.1. Research Objectives 22

1.5.2. Research questions 24

1.8.1. The Advancement Contributed to the Field of Deep Learning 26

1.8.2. Sections of the Thesis that have Undergone Peer Review and been Published 26

1.9. Thesis Structure 28

CHAPTER 2: LITERATURE REVIEW 30

2.1. Exploring the Structure of BERT 30

2.1.1 Pretraining BERT Model 33

2.1.2 Fine Tuning 35

2.2. Role of Transformer Models in Modern NLP. 36

2.3 Model Compression Techniques 38

2.3.1. Pruning Method 38

2.3.2. Quantization 44

2.3.3. Knowledge Distillation 47

2.3.4. Neural Architecture Search (NAS) 51

2.3.4.1. Search Space 51

2.3.4.2. Search Strategy 52

2.3.5. Low-Rank Factorization 55

2.3.5.1. Matrix Decomposition 55

2.3.5.2. Singular Value Decomposition (SVD) 55

2.3.5.3. Fisher-Weighted SVD (FWSVD) 56

2.3.5.4. Progressive Low-Rank Decomposition 57

2.3.5.5. Low-Rank and Sparse Approximation Approach 57

2.3.6. Weight Sharing and Clustering 58
2.3.6.1. Weight Sharing 58
2.3.6.2. Clustering 59
2.3.6.3. Combining Weight Sharing and Clustering 59
2.3.7. Parameter Sharing 60
2.3.7.1. Layer Sharing 60
2.3.7.2. Embedding Sharing 61
2.3.7.3. Grouped Parameter Sharing 61
2.3.8. Tensor Decomposition 62
2.3.8.1. Tucker decomposition 63
2.3.8.2. Canonical Polyadic Decomposition (CPD) 63
2.3.8.3. Tensor Train Decomposition (TTD) 64
2.3.9. Early Exit Mechanisms 65
2.3.9.1. Confidence Estimation 66
2.3.9.2. Internal Ensemble Methods 66
2.3.9.3. Learned Exits 67
2.3.9.4. Global Information Utilization 67
2.4. Applications of Compression Techniques to BERT Models 68
2.4.1. Application of Quantization Techniques to BERT Models 68
2.4.4. Application of Low Rank Factorization 71
2.4.5. Application of Weight Sharing and Clustering 73
2.4.6. Application of Parameter Sharing 74
2.4.7. Application of Tensor Decomposition 75
2.4.8. Application of Early Exit Mechanism 76
2.5. Iterative Magnitude Pruning (IMP) and the Lottery Ticket Hypothesis. 78
2.5.1 IMP 78
2.5.1.3. Advantages and Limitations of IMP 80
2.6 Attention head pruning in BERT. 84
2.6.1. Attention head pruning effects on performance 87
2.6.2. Attention head pruning effects on Model Size 88
2.7 Ensemble Learning Methods in Deep Neural Networks 88

2.7.1 Bagging (Bootstrap Aggregating) 88
2.7.2 Boosting 93
2.7.3 Stacking 98
2.8.1 Role of Subnetworks 101
2.8.2 Impact of Combining Multiple Pruned Models 102
2.9 Sustainability and Resource Efficiency in AI 104
2.9.1 Key Initiatives and Tools for Sustainable AI 104
2.9.2 Challenges and Future Directions 105
2.10. Resource Efficiency in AI 106
2.10.1. Key Strategies for Resource Efficiency 106
2.10.2. Challenges and Future Directions 107
2.11. Research Gaps 108
2.12. The need for advanced pruning framework. 109
Chapter 3: Methodology 113
3.1 Research Paradigm 113
3.2. Research Design 114
3.2.1. Problem Definition and Research Objectives 114
3.3. Proposed Pruning Framework 115
3.3.1. Reviewing Literature on Specific Research Concepts 116
3.3.2. Planning the Review 116
3.3.3. Developing a Review Protocol 116
3.3.4. Conducting the Review 117
3.3.5. Identification of relationships between key concepts 118
3.4 Multi-objective Optimization Algorithm for Performance, Speed, and Resource Efficiency. 119
3.4.1. Efficient Model Pruning Optimization Algorithms 119
3.5 Ensemble of Winning Tickets Method 126
3.5.1. Dataset and Preprocessing 127
3.5.2. Initialization of the Model and Tokenizer 128
3.5.3. Ensemble of Winning Tickets 131
3.5.4. Aggregating Predictions in Ensemble Learning 136
3.5.5. Training Configuration 138

3.5.6. Selection of Informative Attention Heads 138
3.5.8. Performance Metrics 143
3.5.9. Tools and frameworks used for measuring performance. 148
Chapter 4: Results and Discussion 150
4.1. Conceptual Framework Components 150
4.1.1 Sparsification Schemes 150
4.1.4. The Potential Impact of Model Optimization 153
Chapter 5: Conclusions and Recommendations 176
References 179
Appendices 199
Appendix 1: Efficient Model Pruning Optimization Algorithm Code 199
Appendix 2: pruning BERT attention heads_accuracy metrics. 203

LIST OF TABLES

Figure 1: The Transformer model architecture

Figure 2: Pre-training and fine-tuning procedures for BERT

Figure 3: Proposed Conceptual Framework for Pruning of neural networks

Figure 4: Steps of optimizing BERT models through an ensemble of pruned subnetworks

Figure 5: Pruning Percentage Verses Model Size Reduction

Figure 6: Model Size Verses Performance Matrix

Figure 7: Pruning Percentage Verses Performance Matrix

Figure 8: Accuracy and F1 Score Across Pruning Ratios and Steps

Figure 9: Training and validation loss oversteps and pruning ratios

Figure 10: Metrics oversteps: (a) CoLA dataset-F1score and Accuracy scores (b) *SST-2* dataset-F1 and Accuracy scores.

Figure 11: Training and Validation Loss Vs Epochs

Figure 12: Ensemble Iterations and Evaluation Metrics: (a) Ensemble Iterations per Epoch (b) Evaluation Metrics per Epoch

LIST OF FIGURES

Table 1: Network pruning reduces parameters without compromising predictive performance.

Table 2: Performance and Compression Rates of Pruned Models Compared to BERT Baselines

Table 3: Comparison of pruned model with other BERT variants

Table 4: Performance Comparison Between Original and Pruned Models

Table 5: Model performance Comparison with other ensemble techniques.

LIST OF APPENDICES

Appendix 1: Efficient Model Pruning Optimization Algorithm Code

Appendix 2: pruning BERT attention heads accuracy metrics.

ABSTRACT

Large Language Models (LLMs), such as BERT (Bidirectional Encoder Representations from Transformers), have experienced rapid growth and increasing complexity, which has introduced significant challenges related to computational costs, resource consumption, and environmental impact. Although existing pruning techniques offer certain advantages, they frequently entail trade-offs, such as reduced model accuracy, increased computational overhead, and inadequate consideration of sustainability. To address these challenges, in this study we propose a conceptual framework aimed at improving the efficiency and sustainability of BERT models through innovative pruning techniques. This framework incorporates a novel multi-objective optimization algorithm that clarifies the trade-offs among model accuracy, computational expenses, inference speed, and energy usage. Furthermore, the study proposes an Ensemble of Winning Tickets, grounded on the lottery ticket hypothesis, for the pruning of BERT attention heads. Pruning techniques reduce model size while preserving performance, enabling the deployment of highly efficient and environmentally sustainable models. The experimental evaluations show how these techniques are effective to real world tasks including linguistic acceptability and sentiment analysis using the Stanford Sentiment Treebank *(SST-2)* and Corpus of Linguistic Acceptability *(CoLA)* datasets. The results show that the proposed multi objective method achieves significant model compression of BERT Base model being pruned to 70.4 million parameters gaining an accuracy of 96.3% on *SST-2* dataset and F1-score of 95.2%. Furthermore, the pruned model demonstrated F1 score of 82.3% and 56% of Matthews Correlation Coefficient in *CoLA* dataset respectively. The findings of this study also include model size that has been reduced by 36%, with our ensemble model reaching greater performance as compared to the baseline BERT model on both *SST-2* and *CoLA* datasets. The results show F1-score of 94% and 96%, respectively, and accuracy scores of 95% and 96% on the two datasets. The findings of this study demonstrate that advanced pruning techniques enhance model performance, computational efficiency, and environmental sustainability. This study present practical framework to researchers and practitioners to move from large, resource-heavy models to sparse, efficient, and scalable

options, thereby aiding the broader initiative to implement deep learning models in an environmentally conscious way.

Chapter 1: Introduction and Motivation

1.1. The Importance of Model Compression for BERT

Model compression is an important feature for optimizing the deployment of BERT (Bidirectional Encoder Representations from Transformers) Devlin et al. [1] in NLP applications. The BERT model, especially BERT large, has 340 million parameters which are both computationally and resource intensive Lu et al. [2]. The famous large language models with transformer architecture are over parameterized and are in need of lots of resources, like for example BERT as illustrated above have 340 million parameters, OpenAI's Generative Pretraining Transformer (GPT-2) Radford et al. [3] with 1.5 billion parameters, the model of Raffel et al. [4] Text-to-Text Transfer Transformer (T5) has 11 billion parameters and GShard have a massive 600 billion parameters Bianco et al. [5], Sanh et al. [6], Gupta & Agrawal [7]. Several studies have revealed that the baseline models have attained cutting-edge results in several benchmarks [1], making BERT the best Transformer-based model currently available. An interesting investigation made by Strubell et al. [8] indicate that it requires a lot of computational resources which demand substantial high memory consumption and high energy to power this hardware for weeks or even months to train the BERT model.

This complexity creates challenges for real-world deployment, particularly on hardware with limited processing power, such as IoT devices and mobile phones Tambe et al. [9]. To overcome these challenges, model compression techniques like for example knowledge distillation, pruning and quantization, and are used for model size reduction while still achieving high performance Zhao et al. [10]. These strategies do not only make BERT more appropriate for real-life application but also increase its scalability across platforms. This results in another facet of model compression, that is accelerating the inference speed, which will be explained in the subsequent paragraph.

The computational demands of BERT as well as other large models generally suffers from slower inference speeds which is a significant drawback in real-time systems. Because of the

large parameter count, BERT models exhibit long inference times, making them a bottleneck in systems like information retrieval, where response time is highly crucial Wang et al [11]. Model compression techniques are an important factor for enabling this issue by enhancing inference speed. For instance, special protocols such as early exit allow models to provide answers at intermediate stages, which makes the time spent in simple queries for instance shorten Ganesh et al. [12]. This is especially desirable in applications that require quick response times directly influence user experience, like the chatbots and virtual assistants as shown in Yin et al. [13]. In addition to providing an increase of inference speed, techniques of model compression have advantages or benefits in terms of resource efficiency.

Large language models such as BERT, for instance, often requires substantial computational resources, such as high-performance GPUs and memory leading to organizational cost in environments with constrained resources [13]. Because of the increased size and computation complexity of large transformer-based models, they pose challenges to deployment on platforms with minimal memory and computational capacity. Clinical NLP tasks present even greater demands, for example, to facilitate clinical decision-making, where lightweight models are needed to operate effectively in resource-constrained environments Rohanian et al. [14]. For these challenges to be solved, a robust method for compressing models is needed to lower their size and resource requirements. These techniques enable realising of sophisticated NLP capabilities on low resource devices and thus ensuring practical language processing solutions across various applications Min et al. [15]. Thus, in addition to increasing resource efficiency, the impact of model compression extends to environmental considerations.

The growth in the size and complexity of deep learning models has been exponential, and these models contribute significantly to the environmental footprint, due to the high energy consumption required for both training and inference Min et al. [15]. The computational complexities of these large-scale models become less demanding due to model compression techniques, not to mention the conservation of the environment through the reduction of deploying large-scale models. Knowledge distillation and quantization techniques

have been adopted to increase resource efficiency, thus conforming to the Green AI principles that advocates for efficient AI design development Min et al. [15]. Treviso et al. [16] show the effectiveness of optimization strategies, like pruning and quantisation, that can decrease processing demands and energy intake while enhancing sustainability of the advanced NLP models available. In addition, Ramesh et al. [17] stresses that model compression methods do not only increase efficiency but also help in mitigating the environmental costs associated with large models, while maintaining fairness in their application. The focus on resource efficiency and sustainability paves the way for further investigation of compression methodologies into how they can optimize the practical deployment of NLP models.

The compression methods have become more important in different domains such as sentiment analysis, question answering, and information retrieval, where maintaining high accuracy is needed with less computational demands. For instance, the usage of compressed BERT models has shown to achieve a great optimization of the computational overhead as well as retaining accuracy in these tasks Lin et al. [18]. Techniques such as knowledge distillation are useful in this process since they allow the transfer of knowledge of large complex models to small efficient models. The transfer ensures that compressed models are as effective as big models in specific tasks hence useful in deployment in resource-constrained environments. Therefore, model compression does not only improve NLP systems for various applications but also guarantees the scalability and accessible across different platforms of advanced language models.

1.2. Pruning and Its Benefits in Reducing BERT Model Size and Complexity

Pruning is a compression technique whereby redundant layers or individual heads of a large language model can be removed without any degradation in performance and accuracy as demonstrated in Michel et al. [19] Kovaleva et al. [20], Baan et al. [21] and Li et al. [22]. Further to that, some investigations have investigated pruning in the setting of BERT as proposed in the studies of Prasanna et al. [23], Chen et al. [24] and it has proven highly effective in shrinking the storage size of fine-tuned models tailored to specific NLP tasks. Pruning strategies commonly encompass unstructured pruning and structured pruning. In

unstructured pruning, individual weights are eliminated according to their significance, often assessed using their absolute values or gradients Han et al. [25]. The result of unstructured pruning is a sparse model with numerous weights being set to zero. However, while parameters of the model can be significantly reduced, it may not lead to important improvements in inference speed unless specialized hardware's are used Jin et al. [26]. On the contrary, structured pruning involves removing complete components of the model, which might include blocks of weights as suggested by Li et al. [27], attention heads proposed by [19], layers as shown by Fan et al. [28] or even the entire structural parts as presented by Voita et al. [29], McCarley et al. [30] of pre-trained language model.

Reducing the number of parameters is beneficial since it reduces memory footprint of the model and enables quicker inference times that enable real time processing. Studies like that of [31] have investigated pruning during pre-training, the research also revealed that pruning 40% of BERT allows for efficient inference without a significant drop in performance. Further observation from studies shows that pruning together with other methods such as knowledge distillation can retain or even improve the model performance after compression [32]. Pruning the size of complex models, for example, BERT will lead to less energy consumption and have a positive environmental impact [33].

1.3. Challenges and Limitations of Pruning Methods

Pruning methods for large language models (LLMs) can be broadly categorized into unstructured pruning and structured pruning. Both approaches have unique challenges and limitations, especially when applied to transformer-based models like GPT, BERT, and others. Unstructured pruning removes individual weights or connections without considering the overall structure of the model, offering high sparsity but introducing several challenges. One major limitation is hardware inefficiency, as standard processing units like GPUs and TPUs are optimized for dense matrix operations rather than irregular sparsity patterns, leading to minimal speedup despite significant weight reduction [25]. Additionally, unstructured pruning often necessitates retraining or fine-tuning to recover lost accuracy, which is computationally expensive, especially for large models like BERT and GPT-3 Zhu & Gupta [34]. Another key

drawback is the potential loss of important weights, as some pruned connections may play a critical role in model performance, such as capturing long-range dependencies in transformer models Frankle & Carbin [19]. These challenges make unstructured pruning less effective in real-world applications where computational efficiency is a priority.

Structured pruning, on the other hand, removes entire structures such as neurons, layers, or attention heads, leading to more hardware-friendly sparsity patterns but also posing significant challenges. One major issue is the reduction in model capacity, as removing entire structures may limit the expressiveness of the model, negatively affecting performance on complex tasks Michel et al.[19] . Furthermore, structured pruning is highly task-dependent, as the importance of structures varies based on the application, making it difficult to develop universally effective pruning strategies Voita et al. [35]. Lastly, inter-structure dependencies create additional complexity, as removing one structure may alter the function of another, leading to unstable pruning decisions Molchanov et al. [36]. Despite these limitations, structured pruning remains a preferred choice for optimizing models in practical scenarios due to its compatibility with modern hardware accelerators.

Pruning BERT models presents unique challenges, particularly due to the critical role of attention heads in capturing diverse linguistic features and the model's sensitivity to task-specific requirements. For instance, Michel et al. [19] demonstrated that while many attention heads in BERT are redundant, pruning them indiscriminately can degrade performance on tasks requiring specialized heads, such as coreference resolution or syntactic parsing. Additionally, task-specific sensitivity complicates pruning, as heads important for one task (e.g., sentiment analysis) may be less relevant for another (e.g., question answering), as highlighted by Voita et al. [35] . Furthermore, inter-head dependencies make it difficult to assess redundancy in isolation, as pruning one head may increase the importance of another, leading to unstable pruning decisions Molchanov et al. [36].

Our research directly addresses these challenges by introducing a conceptual framework that enhances BERT’s efficiency through advanced pruning techniques while maintaining its performance across NLP tasks. First, we analyze the shortcomings of existing

pruning methods to identify key areas for improvement regarding computational cost, accuracy loss, and sustainability. To overcome the trade-offs between model accuracy, inference speed, and resource consumption, we propose a novel multi-objective optimization algorithm that dynamically balances these factors during the pruning process. Additionally, we introduce the Ensemble of Winning Tickets approach, which selects and fine-tunes optimal subsets of pruned attention heads, preserving essential linguistic features and enhancing model robustness. This method mitigates the instability of structured pruning by leveraging multiple pruned configurations, ensuring that no critical structures are lost. Finally, we validate our framework through extensive evaluation of NLP tasks which are sentiment classification and linguistic acceptability, comparing the performance of our pruned models against existing benchmarks.

1.4. Statement of Problem

The growing complexity of BERT as a large language model, although very important for NLP tasks, there are significant challenges related to computational expenses, resource consumption, and environmental impact. Jaiswal et al. [37] reveal that large language models are computationally expensive, and excessive memory demands of large models, highlighting the necessity for pruning techniques to address these challenges. While the existing pruning strategies are beneficial and comprehensive, they often lead to trade-offs such as performance degradation and increased computational overhead as highlighted in Wang et al. [38]. These models often do not achieve the optimal balance of resource savings with model accuracy, particularly in real-world applications. Furthermore, at present, many of the current pruning methods lack focus on sustainability; for instance, the study on enhancing clinical language [39] highlights the negative environmental impact of energy-intensive computations, stating the critical need for greener, more efficient pruning strategies. Determining optimal pruning criteria is another prevailing challenge, as noted in various studies-which discuss the importance of balancing performance with resource savings.

These shortcomings lay the groundwork for this research through the proposal of a conceptual framework. for enhancing BERT model efficiency through advanced pruning

techniques. Particularly, our study presents a multi-objective optimization algorithm that formalizes the balancing between computational cost, accuracy, inference speed, and energy consumption. Additionally, the research proposes a novel method of pruning BERT attention heads based on an Ensemble of Winning Tickets considered under the lottery ticket hypothesis which maintains and enhances the model's performance while simplifying it. The proposed research tries to go further and tackle both computational and sustainability issues to provide the basis for more practical, efficient, and environmentally responsible deployment of BERT models across various real-world applications.

1.5. Main Objective

The goal of this study is to formulate a conceptual framework that enhances BERT's computational efficiency and environmental sustainability through advanced pruning techniques. While this framework is validated on BERT, its principles such as the multi-objective optimization algorithm and the Ensemble of Winning Tickets approach are designed to generalize to other transformer-based architectures (e.g., GPT, T5, and encoder-decoder models). This paves the way for future research in two key directions: (1) hardware-aware pruning strategies tailored to edge devices (e.g., IoT, mobile GPUs), where memory and energy constraints are critical, and (2) dynamic compression techniques that adapt to model architectures evolving toward larger scales (e.g., trillion-parameter models). By addressing these gaps, this study contributes a foundation for scalable and sustainable deployment of NLP systems across domains.

1.5.1. Research Objectives

1. **To analyse the shortfalls in existing pruning frameworks** (Unstructured pruning trade-offs in accuracy and efficiency)
   Building on the limitations identified by Wang et al. [38] and Michel et al. [19], this objective systematically evaluates why current pruning methods fail to balance accuracy loss e.g., task-specific head removal [20] with computational gains e.g., GPU latency [26].

2. **To propose a novel multi-objective optimization algorithm** (Lack of holistic optimization)

   Directly responding to the hardware inefficiency challenges noted in Section 1.4 [25], [26], this objective formalizes a multi-objective optimization algorithm that simultaneously optimizes:

   - Model accuracy (mitigating performance degradation risks [38])
   - Inference speed (targeting GPU/TPU architectures [26])
   - Energy consumption (addressing environmental concerns [15])

3. **To propose an Ensemble of Winning Tickets method** (Instability in structured pruning)

   In contrast to the task-sensitive pruning limitations highlighted by Voita et al. [35] and Molchanov et al. [36], this objective develops a robust attention-head selection strategy that preserves linguistic capabilities while reducing model size by ≥40% [19], [23].

4. **To validate the proposed framework** (Insufficient real-world benchmarking)

   Where prior studies focus narrowly on accuracy metrics (e.g., [24]), this objective evaluates:

   - Computational efficiency (FLOPs reduction vs. [25])
   - Environmental impact (energy savings vs. [15])
   - Task performance (sentiment analysis, linguistic acceptability vs. [13], [18]

5. **To measure cross-task impact** (Limited sustainability focus)

   Extending beyond conventional accuracy benchmarks (e.g., GLUE), this objective quantifies:

   - Energy efficiency gains (watts per inference vs. [8], [15])
   - Hardware deployment potential (edge-device compatibility vs. [9], [14])

1.5.2. Research questions

1. What are the limitations and gaps in current pruning techniques for BERT models, particularly in terms of computational overhead, accuracy loss, and sustainability challenges?
2. How can a multi-objective optimization algorithm be designed to strike a balance between maintaining model accuracy, reducing computational overhead, improving inference speed, and minimizing energy consumption in the pruning process?
3. How does the use of an Ensemble of Winning Tickets for pruning attention heads in BERT impact model complexity and performance, and how does it compare to existing pruning methods?
4. What are the effects of the proposed pruning techniques on the computational efficiency and energy consumption?
5. How does the proposed pruning method affect the efficiency and accuracy of compressed BERT models in NLP applications such as sentiment analysis and linguistic acceptability?

1.6. Justification of the Study

The size and the increasing intricacy of LLMs such as BERT significantly affect computation and hinder their use in low-power computing scenarios such as smartphones and embedded IoT systems. BERT is a popular transformer-based model, which carries millions of parameters making it impossible to implement in real-time due to large memory requirements. These studies by [12] and Rogers et al. [40] shows that deploying large language models for sentiment analysis involves significant computational costs, largely due to the model's size and complexity, indicating that efficient deployment solutions are needed. This PhD study seeks to address these issues through advanced pruning techniques that reduce computational overhead without compromising model performance.

Further, the environmental sustainability of artificial intelligence (AI) is becoming a vital concern. Large AI models training and deployment have an impact on the constant rise in energy use and CO2 emissions. Such work as Xu and McAuley's [41] does point to the

environmental impacts of AI and demands for more energy-saving approaches. This research is particularly justified as it seeks to focus on balancing computational efficiency and sustainability. By introducing novel pruning methods, this paper aims to minimize energy and carbon cost of large model use, contributing to Green AI practices.

1.7. Significance of Study

This research offers a novel contribution to model compression and sustainability by proposing a novel framework for increasing the computational efficiency of large-scale language models through advanced pruning techniques. In particular, this study proposes a multi-objective optimization algorithm that formalizes the balance between computational cost, accuracy, inference speed, and energy consumption. This represents a key aspect that adds towards the optimization of LLMs for both computational efficiency and sustainability. Additionally, the research proposes a novel method of pruning BERT attention heads based on an Ensemble of Winning Tickets considered under the lottery ticket hypothesis which maintains and enhances the model's performance while simplifying it.

The efficient pruning techniques, like most of the studies have suggested, allow for faster and more resource-efficient models, which are necessary for tasks like language translation and decision making. In addition, this research also addresses gaps for deploying AI in resource-constrained environments, which possibly could encourage democratizing access to AI technology in developing countries and low-cost devices. In this way, the findings of this work will significantly contribute to more practical and sustainable AI deployment, this study will have a lasting impact on the field of AI research.

1.8. Contributions

The contributions of this PhD research are presented in two forms: The contributions include the body of knowledge in deep learning and published works generated as a result of this study

1.8.1. The Advancement Contributed to the Field of Deep Learning

We present a conceptual framework alongside a new multi-objective optimization algorithm that balances pruning decisions across accuracy, computational cost, inference speed, and energy consumption. Through analyzing the connections between these concepts and assessing computational efficiency and power consumption, the framework and the novel multi-objective optimization algorithm goal is to elaborate practical steps to enable efficient resource usage and energy expenses in real-world NLP projects using pruning strategies. This work aims to look for a balance between the size of the models and their accuracy and, also to provide green directions in Natural Language Processing. This framework, along with the algorithm, functions as a valuable reference for researchers and experts, who can select a suitable approach based on the specific application requirements for deep learning pruning.

We also introduce a new pruning technique that we call Ensemble of Winning Tickets which utilizes the lottery ticket hypothesis to enhance BERT's performance metrics while drastically reducing its complexity. In this proposed approach, we implement an iterative attention head pruning for BERT retaining the minimal features of the model. In our approach the combined pruned subnetworks or winning tickets are combined into a robust ensemble model which gives up to 36% reduction in model size without deteriorating the performance. Pruning techniques have become important in overcoming the challenges of increasing model complexity in deep neural networks. The techniques of pruning streamlines network structures, improving inference speed, reducing memory usage, and enhancing computational efficiency by selectively removing redundant or less impactful components. A promising approach is the concept of "winning tickets," which identifies efficient subnetworks within sparse deep neural networks that match or exceed the performance of the original model.

1.8.2. Sections of the Thesis that have Undergone Peer Review and been Published

1. our original research article titled *"Ensemble of Winning Tickets: Pruning BERT Attention Heads for Enhanced Model Efficiency"* Smarts et al. [42] Published in the International Journal of Electrical and Computer Engineering (IJECE), p-ISSN 2088-8708, e-ISSN 2722-2578, a Scopus/ScimagoJR indexed journal, DOI:

http://doi.org/10.11591/ijece.v15i2.pp2070-2080. This paper was published as one of our objectives from the thesis whereby we introduced a new pruning technique that we call Ensemble of Winning Tickets which utilizes the lottery ticket hypothesis. The approach implements an iterative attention head pruning for BERT retaining the minimal features of the model.

2. The review paper titled *"Natural Language Processing Revolution from Humble Beginnings to Deep Learning and Beyond"*, Presented to the 4th International Conference on ICT in Business, Industry and Government, the flagship conference of IEEE Computer Society, the paper published by IEEE Xplore, DOI: 10.1109/ICTBIG64922.2024.10911728. Our literature review from the thesis was prepared into a review paper which was peer reviewed and published by IEEE Xplore. The paper highlights key NLP concepts, tracing chronological advancements, exploring deep learning innovations, and discussing effective pruning techniques to optimize model performance. The paper also suggests future developments for scholars to investigate and find solutions for current challenges in NLP. Its results not only act as a guide for practitioners today but also suggest new directions for research and development that will help meet the challenges of tomorrow in this dynamic discipline of NLP.

3. An original research article titled *"A Conceptual Framework for Efficient and Sustainable Pruning Techniques in Deep Learning Models.* Published by the Journal of Computer Science". Could be cited as: Smarts, N., Selvaraj , R. & Kuthadi, V. (2025). A Conceptual Framework for Efficient and Sustainable Pruning Techniques in Deep Learning Models. Journal of Computer Science, 21(5), 1113-1128. https://doi.org/10.3844/jcssp.2025.1113.1128. The other objective of this thesis is to introduce a conceptual framework and a novel multi-objective optimization algorithm that formalizes pruning as a balance between computational cost, accuracy, inference speed, and energy consumption. The objective was prepared into an original research paper which was peer reviewed and published. The paper proposes a conceptual framework that emphasizes sparsity-driven efficiency, it discusses how it contributes to the realization of sustainable technological development, fostering the development of environmentally friendly and

sustainable technologies. The paper also discusses how the framework was developed in the following steps: 1) reviewing the literature on key research concepts which are model pruning, efficiency, inference, and sustainable technology development. 2) identifying connections between research concepts. 3) design and development of the proposed conceptual framework for pruning. The paper also discusses the novel multi-objective optimization algorithm that prunes the model while maintaining accuracy, reducing computational cost, minimizing inference time, and optimizing energy consumption

1.9. Thesis Structure

This thesis is further divided as follows: Chapter 2 introduces the structure of BERT, and how transformer-based models are significant to NLP, and continues to discuss the model compression techniques such as knowledge distillation, quantization and pruning with a specific focus to BERT. Further, the critical pruning approaches, such as Iterative Magnitude Pruning (IMP), and attention head pruning are discussed and along with ensemble methods, wherein the pruned subnetworks are combined for enhanced performance. Issues related to sustainability in Artificial Intelligence and highlighting Green AI practices for decreasing energy consumption are explored. Finally, it highlights some of the gaps present in the current techniques for pruning BERT and ensemble methods; this forms the motivation for an advanced pruning framework.

Chapter 3 outlines the Methodology and Theoretical Framework of the study. Starting with the general research design of the investigation, followed by an in-depth explanation of the proposed pruning framework for pruning BERT's attention heads through a multi-objective approach balancing performance, speed, and resource efficiency. This current chapter presents the idea of ensemble of winning tickets; it discusses the process of subnetworks (winning tickets) selection and how they are trained to create an efficient model ensemble. The dataset used as well as the preprocessing and tokenization processes are being explained, it also details the training and pruning of BERT using the iterative magnitude pruning technique. The chapter closes by discussing the following metrics accuracy and inference speed, memory usage and energy consumption including the tools for evaluation.

In Chapter 4 the results of the research together with the comprehensive discussion of their implications are presented here. The outcomes are structured in such a way that they shed light on the baseline results of the unpruned BERT model and the outcomes of applying cutting-edge pruning techniques, and the benefits of combining pruned subnetworks into an ensemble. Additionally, the chapter delves into resource efficiency, resource utilization, and the contributions of the proposed framework to sustainable AI. Chapter 5 brings the thesis to a close by outlining the main discoveries, with a particular focus on the study's impact on the fields of model compression and neural ensemble strategies, notably for BERT models. It does sum up the theoretical progress practical advancements made in pruning, while pointing out limitations related to trade-offs and generalization challenges. Recommendations for future research consist of expanding the advanced pruning strategies and applying these techniques to other deep learning models and architectures paving the way for further improvement of efficient AI methods.

CHAPTER 2: LITERATURE REVIEW

2.1. Exploring the Structure of BERT

BERT has enhanced the field of NLP because of its novel architecture, the model which was introduced by [1], employs bidirectional methodology for modeling a language since it is able to determine left context as well as right context at the same time, it is different from previous unidirectional approaches [12]. Bidirectional processing is implemented using the masked language modeling (MLM) approach and involves masking and training the model on randomly selected words in a given sentence Lai et al. [43] based on their surrounding context.

The benefit of BERT architecture BERT is that it was built upon the transformer architecture originally introduced by Vaswani et al. [44]. A study of [44] have indicated the Transformer model leverages self-attention to assign contextual importance to different words in a sentence. The self-attention mechanism captures complex relationships and dependencies as compared to the limitations associated with the recurrent neural networks (RNNs) Hernández and Amigó [45]. BERT is made up of multiple layers of encoders, the multi-head self-attention mechanisms in every layer empower the model to process and emphasize various components of the input sequence concurrently [16]. According to the study by Roy et al. [46] the architecture does not only enhance the model's contextual understanding but also enables parallelization during training, making it more efficient than traditional sequential models.

Here is an example which demonstrate bidirectional training approach, a sentence “The dog ran to the chicken,” the model realize that “dog” and “chicken” are related given the context of the entire sentence, not just prior words. The power of being able to comprehend words in context is valuable for cases like question answering and natural language inference where getting the whole context is important for making the right prediction [11]. This is also true, as emphasised by Hou et al. [47] further highlights that it enables fine-tuning on specific tasks with little alterations allowing it to serve as a highly adaptable tool across diverse NLP

applications. BERT's innovative use of the Transformer architecture and its bidirectional training approach have significantly raised the standards in NLP, demonstrating important improvements across multiple tasks and establishing it as a foundational model for subsequent research and applications in the field Campino [48].

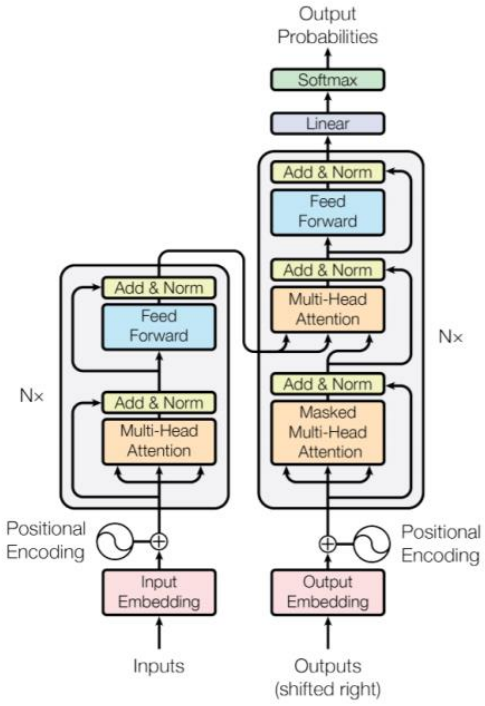


Figure 1: The Transformer model architecture [44]

The above figure 1 illustrates the transformer neural network architecture that was initially created to solve the problem of language translation. Researchers and practitioners have very well utilized it, before the inception of the transformers the Long Short Term Memory networks (LSTM) Gers and Schmidhuber [49] and Hochreiter and Schmidhuber [50] were used to solve this problem but experience some limitations. Wang and Jiang, [51] in their study they reflect that the premise and the hypothesis is decreased to a single embedding vector before matching them. LSTM networks have slow training times due to their sequential processing of words, both in input and output, requiring a large number of timestamps for effective learning. The models are not the best option when capturing the true meaning of words. Even the bidirectional-LSTMs Graves and Schmidhuber [52] they process

left-right and right-left contexts in isolation before concatenating them, leading to a partial loss of the original context. The transformer model tackles these limitations effectively by processing words concurrently, resulting in much faster performance. Their deeply bidirectional nature allows them to capture word context more effectively by processing information from both directions at the same time.

At the core of the transformer are its two primary elements, the encoder and decoder, crucial for understanding its operation, let's give an example of training this architecture to convert English to Setswana language. The encoder handles the English words in parallel, generating embeddings for each word at the same time. The embeddings are vector representations that capture word meanings, and words with similar meanings have vectors that are numerically closer. The decoder takes the encoder's embeddings and the preceding words of the Setswana translation to sequentially generate the next word in the output sentence. The translation into Setswana is built sequentially, one word at a time, until the entire sentence is formed. This approach is more appealing than some LSTMs because it distinctly separates tasks, with the encoder focusing on understanding English, grammar, and, most importantly, context. The decoder, on the other hand, learns the connections and mappings between English and Setswana words. Even individually, both parts have a deep underlying knowledge of language, allowing sections of this architecture to be adapted for developing systems that understand language.

Stacking the decoder part of transformer OpenAI's GPT will be created and conversely when the encoders component is stacked, we get BERT Ghaemi et al. [53]. BERT can be utilized for learning neural machine translation, answering questions, analyzing sentiment, and summarizing text and many more tasks Lamsiyah et al. [54]. All these tasks need the understanding of language, therefore BERT is first trained to grasp language and can later be adapted through fine-tuning for particular problem-solving needs. BERT is trained in two phases, starting with pretraining, where the model acquires knowledge of language and contextual relationships. In the fine-tuning phase, the model is trained to grasp the

relationships and interactions between two sentences Su and Vijay-Shanker [55]. The pre-training and fine tuning process is illustrated in figure 2 below.

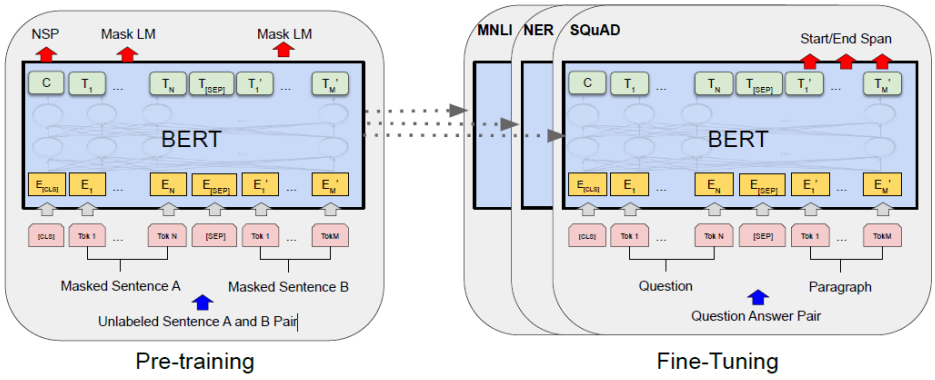


Figure 2: pre-training and fine-tuning procedures for BERT [1]

2.1.1 Pretraining BERT Model

The purpose of pretraining is to enable BERT to understand the fundamentals of language and the concept of context. Liu [56]. To develop linguistic proficiency, BERT is trained concurrently on two key unsupervised objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) [40], Kryeziu and Shehu [57]. The MLM task requires BERT to reconstruct masked words within a sentence, enabling it to develop a deeper understanding of bidirectional linguistic dependencies. For NSP, BERT receives a pair of sentences and predicts whether the second sentence is a natural continuation of the first, using a binary classification approach. This enables BERT to grasp contextual relationships between sentences, and by combining both tasks, it develops a deeper understanding of language Liu et al. [58].

During pretraining, BERT is trained in MLM and NSP, but in implementation, both tasks are learned concurrently. The model receives two sentences as input, with some words hidden, and each token corresponds to a word, and those words are converted into embedding using pretrained embeddings. The process could be represented mathematically, like on the following notations and equations:

1. Input Representation: Let S_A and S_B be two sentences, where:

$$S_A = \{w_{A1}, w_{A2}, \dots, w_{AN}\} \quad (1)$$

$$S_B = \{w_{B1}, w_{B2}, \dots, w_{BM}\} \quad (2)$$

Here, N and M indicate the number of words in sentences A and B, respectively.

2. Masked Language Modeling ($\boldsymbol{MLM}$): Let M Denote the group of positions where words have been masked in S_A and S_B. The masked input can be represented as:

$$\tilde{S} = \{w_{A1}, w_{A2}, \dots, w_{AM}, \text{MASK} , \dots , \text{MASK} \} \quad (3)$$

where the words at indices in M are replaced with a special token $MASK$.

3. Embedding Representation: Each token w_{Ai} and w_{Bi} is converted into an embedding vector using a pre-trained embedding function E: $\mathbf{E}(w_{Ai})$ and $\mathbf{E}(w_{Bi})$

The input embeddings are represented as:

$$\mathbf{X} = \{\mathbf{E}(w_{A1}), \mathbf{E}(w_{A2}), \dots, \mathbf{E}(w_{AN}), \mathbf{E}(\text{MASK}), \dots\} \quad (4)$$

4. Next Sentence Prediction: The output C for the next sentence prediction can be defined as:

$$C = \begin{cases} 1 & \text{if } S_B \text{ follows } S_A \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

5. Masked Language Modeling Output: Let $\mathbf{T} = \{\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_K\}$ be the set of output word vectors linked to the hidden words, where K is the number of masked tokens. The output for the MLM problem can be represented as:

$$\mathbf{T}_i = \text{MLM}(\tilde{S}) \quad (6)$$

where MLM is the function that predicts the original tokens for the masked positions.

6. Input-Output Relationship: The total input token vectors correspond directly to the total output token vectors:

$$|\mathbf{X}| = |\mathbf{T}| \tag{7}$$

To illustrate the process of applying MLM and NSP to transformer-based models like BERT is being shown by the above mathematical framework. In the first place the input representations, embeddings are created for each token, including masked positions, to create a more contextual aware embedding sequence. On the other hand, the MLM function is then to predicts the masked tokens, while the NSP task determines the consistency between sentence pairs. By maintaining a consistent input-output relationship, this structure ensures effective model training, enabling both performance enhancement and flexibility in various NLP applications.

On the input side, the first-layer representation is built using token embeddings, segment embeddings, and position embeddings. The token embeddings are the pretrained embeddings, [1] which use the word piece embeddings as indicated by Gomez-Perez et al. [59] comprising a vocabulary set of 30k tokens. The segment embedding represents the sentence index encoded as a vector, while the position embedding captures the word's location within the sentence in vector form. By summing these three vectors, an embedding vector is formed, which serves as the input to BERT. Since all vectors enter BERT at once, segment and position embeddings are necessary to preserve the temporal sequence required for effective language modeling.

2.1.2 Fine Tuning

Adapting a BERT model through fine-tuning is essential for improving its performance on NLP tasks. To demonstrate the process, start with a pre-trained BERT model, which has acquired general language representations from an extensive corpus, and is initialized with weights that encompass a wide range of linguistic features Wang et al. [60]. To adapt BERT for particular tasks like named entity recognition, sentiment analysis and question answering a task-specific output layer is added. The layer be denoted as $f(x; \theta, \gamma)$, where x is the

input, θ represents the model parameters, and γ denotes the parameters of the new output layer. The fine-tuning process adapts the model by training it on a labeled dataset relevant to the specific application, minimizing a loss function $L(\theta, \gamma)$ over the dataset D:

$$L(\theta, \gamma) = \frac{1}{|D|} \sum_{(x_i, y_i) \in D} \mathcal{L}(f(x_i; \theta, \gamma), y_i) \quad (8)$$

where y_i is the true output for x_i whereas $\mathcal{L}$ represents a loss function, like cross-entropy. As the BERT model goes through fine-tuning it becomes task aware, thus improving its performance as compared to utilizing a pre-trained model. The process is mostly important in tasks such as question and answering, Allowing the model to determine the exact start and end locations of an answer span in a contextual paragraph, represented by (s, e), where s and e are the start and end indices, respectively. Consequently, fine-tuning enables BERT's effective application across various NLP tasks, improving both accuracy and efficiency in language processing. As indicated earlier on, BERT is formed when layers of encoders stacked together. The key difference between BERT Base and BERT Large lies in the number of encoder layers. BERT Base is built with 12 encoder layers, while BERT Large doubles this with 24 encoder layers Ganesh et al. [61].

2.2. Role of Transformer Models in Modern NLP.

As studies have suggested, the landscape of NLP has been drastically changed by the transformer models due to their novel architecture that has importantly improved the ability for language generation and understanding. The model first proposed by [44] as earlier indicated, leverages self-attention to weigh the importance of different words in a sentence, enabling it to model complex relationships and dependencies while avoiding the limitations of sequential processing in recurrent neural network (RNNs) Torfi et al. [62]. This ability enables the transformers Handles entire text sequences at once, making it highly effective for tasks that demand long-range contextual understanding, such as question answering, text summarization and machine translation.

The creation of pre-trained language models such as BERT and GPT are emerging as one of the most significant developments in NLP. According to Raiaan et al. [63] the ability to be bidirectional greatly enhances performance on various NLP tasks which involve a deep appreciation of language intricacies; this is because the model can comprehend words in their entirety by considering the full sentence context. In the example of “The bank can deny lending money “the BERT model is able to infer from the context that the term “bank” refers to a financial entity and not the edge of a river.

On the other hand, the GPT models including GPT-2, GPT-3 use a unidirectional method, where the network predicts the next word in a sentence based on the words that come before it. Nonetheless, the models have shown excellent performance in producing syntactic and semantically meaningful, operationally relevant text, and set new records in tasks like text completion and dialogue generation [60]. These models are highly effective as a result of being trained on large-scale data which allows them to learn rich language representations and semantic knowledge.

Transformer models are not limited to language processing only; they have been effectively utilized across various domains. of activity, like in healthcare, where they can contribute to clinical decision-making, by analyzing patient data and predicting outcomes [39]. For instance, the DPAL-BERT model used for question-answering for clinical practice improves the preliminary BERT design in terms of enhancing efficiency and performance while minimizing computational needs, making it ideal for use in resource-limited settings [13]. In addition, the transformer architecture has open doors for innovations in model compression methods designed to shrink the size and complexity of models while maintaining performance. Techniques like quantization, knowledge distillation and pruning have been employed in order to make large models like GPT and BERT more efficient and accessible for real-world applications [7]. This is definitely useful, especially as we observe a rising trend in the need to run NLP models on mobile and other edge devices. Transformer models are critically essential for NLP since they advance the comprehension and generation of language and lay the foundations for improved and more diverse solutions in multiple domains. Its

ability to learn and model complex language patterns has brought about massive enhancements in tackling NL innovative language understanding problems making them the building blocks of current language processing technologies Wolf et al. [64].

2.3 Model Compression Techniques

2.3.1. Pruning Method

It is a model compression method whereby redundant layers or individual heads of BERT can be removed without any degradation in performance [19], [20]; Baan et al., [65] and accuracy [22]. Further to that, some investigation have looked into pruning in the setting of BERT Chen et al.[66] and It has demonstrated significant success in reducing the storage requirements of models tailored for specific NLP tasks. Due to the reduced memory size, large models can be implemented on edge computing devices such as edge devices, mobile phones and sensors. There are two main categories where pruning fall into, which are structured and unstructured pruning [22].

2.3.1.1. Structured Pruning

Noticeably some studies have discovered that structured pruning mostly leads to a reduction in inference cost and model size. In addition to that, structured pruning can be used to remove blocks of weights [22] attention heads [19], layers [28] or even the entire structural parts [30], Voita et al. [35] of pre-trained language model, to ensure that it is smaller in size and less complicated. The first technique of pruning pretrained language models that we shall discuss is block structure pruning. The study by [22] proposed using block structured pruning in pretrained language representation, whereby a reweighted group Lasso was incorporated for optimization. Their work have its footprint on that of Wen et al. [67], they have also demonstrated the use of group Lasso acts as a relaxation of the hard constraints, thus making the transformer-based model become more capable and performs well on computation and remains accurate.

Secondly, pruning attention heads can remarkably have the model performing well and giving higher accuracy with only a few attention heads per encoder unit, at the same time, it is important to acknowledge that the BERT baseline model runs on sixteen heads [19]. Furthermore, the study illustrate that a single head is enough for training at test time using twelve (BERT) attention heads. It is worth noting that Zhang et al. [67] introduced the Single-Shot Meta-Pruning method for eliminating attention heads from deep pre-trained Transformers, the pruning took place before fine tuning because the study states that it helps with reducing computational overhead during the stages of fine-tuning and inference. Moreover the study by [35] illustrated an important discovery through identification of important specific attention heads in the encoder and how they affect the performance of the model. The study went further and examines the role played by each individual attention head. The method for identifying the usefulness and the role played by each specific attention head was enabled by the use of stochastic gates combined with a differentiable relaxation of the L0 penalty. Another work proposed a pruning technique which they termed differentiable subset pruning Li et al.[68] sparsifying Transformers. The advantage of this technique is that it can enable users to establish the level of sparsity within a model.

In their paper Lin et al. [69] revealed a pruning method that wholly penalizes the sub-modules which might include; attention heads, feed-forward sub-networks, attention modules etc. The paper reports that in order to conduct an effective pruning, spectral regularization must be utilized and also to achieve good model compression pruning must be applied on mappings. The paper by Parnami et al. [70] used A* search algorithm to aid in compressing the transformer model and remove redundant heads for them to achieve higher accuracy. Compared to Kim et al., [71] the paper by Wang et al. [72] demonstrated what they have termed (spatten), a cascade token and head pruning method that have shown efficiency in removing tokens and heads from layers. In contrast [71] proposed a token pruning method which operates in an adaptive manner as input sequence passes through layers it gets rid of insignificant tokens. The value of the attention score is learnt during training and those whose scores are below the threshold goes through pruning. Furthermore [72] in their paper came up with three optimizing algorithm to cut down computation and memory access which are

cascade head pruning, progressive quantization and cascade token pruning. Local value (V) pruning is also introduced by [72] whereby pruning happens after softmax and the current head's probabilities identify the V vectors to be removed.

Khetan and Karnin [73] explored pruning-based architecture search technique by parameterizing each layer with five different dimensions, then re-train each variant and apply pruning-based architecture search technique to optimize the model. Similarly, [1] using the same technique suggested a fixed value for other dimensions and parameterizes the others. In the same way Liu et al. [74], Pham et al. [75] , Singh et al. [76] have applied Neural Architecture Search (NAS). Likewise, Tsai et al. [77] re-parameterized the transformer model and at the same time search the model's depth and width using one-shot architecture search method for optimal inference. An interesting aspect to point out Sridhar and Sarah [78] have emphasized what is call the intermediate layers within the BERT model and their work committed to look into the impact of those layers on how the network is performing in general. The study demonstrated that pruning the intermediate layers and changing the structure of BERT had little impact on accuracy. In the same order Sajjad et al. [79] examined how to remove top, bottom, middle, alternate, or layers that contribute the least to the network. According to the findings, many attention heads can be removed during testing without compromising performance [19], regardless of models being trained with many heads. They further illustrate unbelievably that a model can have many layers can be converted to a single attention head without negatively reducing the performance of the model during test time, but that most of heads must stay intact to avoid disastrous drops in performance. The performance of the winning tickets can be at par or overboard compared to that of the original model Liang et al. [80] if appropriate compression ratios are applied. The study focused on the slightly compressed sub-networks and how they operate within large language models

Another technique for pruning of transformer network is layer dropout as it is applied by [28] in what they have termed layer-Drop it doesn't require any fine tuning and smaller networks of a chosen complexity are selected from the large network with no interference on performance. In layer dropout a plan is devised in advance during training to randomly drop

encoder units like dropping model weights in Dropout Hinton et al. [81] or DropConnect Wan et al. [82]. Previous work Huang et al. [83] has worked in layer dropout on deep convolutional networks and came to a conclusion that it can prevent over-fitting in neural networks and shorten training time.

2.3.1.2. Unstructured Pruning

The other category of pruning individual weights that we shall discuss is a paradigm called unstructured or sparse pruning. It can be considered to be one of the best ways to compress a network by identifying the less important weights in the model; therefore the least important fine grained connections in the over-parameterized model are individually removed from the network by adjusting them to zero. Early work can be traced back to LeCun et al. [84] and Hassibi et al. [85] where they used a second-order Taylor expansion approximation to estimate the target and predicted output values of a trained model. However, unstructured pruning often results in most frameworks and hardware failing to accelerate sparse matrices, which means that it cannot be utilized without computing resources or libraries available exclusively for the task at hand [9] and [25]. However, the networks can be pruned in such a way that architecture of the network is directly changed, which any framework can handle. We will cover different techniques or criteria used by researchers and practitioners when applying unstructured pruning to pretrained BERT model. There are a number of unstructured criteria or techniques used in the literature like; magnitude weight pruning, Movement pruning and reweighted proximal pruning etc.

On magnitude weight pruning weights that are far from zero are considered, [66], Gordon et al. [86], Mao et al. [87] this means that if the weight values are greater than zero they remain, if they are zero then just splash them out. In contrast to other works which discovered their sub networks after some training or fine-tuning [66] found theirs at (pretrained) initialization and in terms of performance, they surpassed those derived by arbitrarily pruning and arbitrarily reinitializing. The method only prunes the model without fine-tuning Jiang et al. [88] making pruning decisions using movement trend of weights not

the actual movement in movement pruning. For further enhancement of the method a new masking function was introduced to align the remaining weights with the architecture of the (PLMs). A recent study Huang et al. [89] created a framework that automatically prunes weights, despite the overall pruning ratio constraints; it identifies a strong sub-network candidate with excellent accuracy. It eliminates the need for expert knowledge and extensive training iterations, relying instead on available auto-weight pruning techniques while preserving accuracy He et al. [90], Liu et al. [91]. Studies have revealed that magnitude pruning works well with regularization $(L1/L2)$ while training and to achieve effective pruning dropouts can also be utilized.

As stipulated in the paper by [6] movement pruning is a first-order approach that relies on fine-tuning objectives [9]. Something that we should note about movement pruning is that weights that have low and high values, if they shrink during training they can be eradicated [6]. Furthermore, movement pruning is very effective when compressing models for transfer learning. Soft movement pruning introduced by [6], on the other hand, is a type of movement pruning that is centered on the binary mask function Mallya and Lazebnik [92], [93]. A recent study Zhang et al. [93] presents reinforced evolutionary search as a method for determining the optimal layer-wise sparsity that meets the target latency requirement while preserving accuracy, using structured based movement pruning rather than magnitude pruning. Guo et al. [94] inspired by the experiments done and Loshchilov and Hutter [95] they developed Reweighted Proximal Pruning (RPP), which integrates reweighted L1 minimization Candes et al. [96] and the proximal algorithm Parikh et al. [97] for large scale models. The method was the first to achieve an actual weight pruning result on language models like BERT.

2.3.1.3. Hybrid Pruning

Hybrid pruning involves a combination of methods which are used to remove individual weights, neurons, attention heads, layers, weight blocks, or even entire structural parts as examined by [22]. LadaBERT, a combination of knowledge distillation, matrix factorization, and weight pruning, was proposed by [87]. According to the study, the

researchers took advantage of combining different compression methods and applied weight pruning to matrix factorization using Singular Value Decomposition (SVD) matrices. Zafrir et al. [98]combined weight pruning and model distillation to produce a method for training sparse language models that can keep the sparsity pattern while transferring learning to new tasks. The method introduced by Kurtic et al. [99] employs a blend of second order pruning and block pruning of weights that have been scaled for the first time to BERT and it's also highly accurate. The method being second order approach was adopted from the Optimal Brain Surgeon framework [23], [84]. The study by [23] have outlined the position of attention heads during pruning that the ones trimmed at the end tend to assume the first and middle layers. It is very clear from their observations that whether good or bad sub-networks may be fine-tuned to perform well after pruning, so most BERT weights are usually useful. The work of [23] found its inspiration from [66] and [86] because they also used magnitude weight pruning, implying that the pruned model's weights, prior to fine-tuning, can achieve performance comparable to the original model. At the fine-tuning stage, Lagunas et al. [100] approach integrates block pruning and movement pruning techniques, and it provides the ability to prune full model components.

Furthermore, because the proposed method by Fan et al. [101] avoids irregular memory access, functions globally based on signals from the top layer, and each layer is pruned by spreading the output of global signals through layers, the proposed model of MI-based layer-wise pruning results in a high speed up of the model with the same sparsity. Xia and shin [102] discovered that element wise and rank pruning strategies can over all tasks outperform training models separately because of the multitask objective. They experimented with various pruning strategies, such as magnitude and movement, element-wise pruning and rank pruning and global with local which hybridized with one another. Kim and Awadalla [103] to significantly improve the model's inference efficiency, they employed Knowledge Distillation, Structured Pruning and Model Quantization. On model pruning they focused on reducing attention heads and feedforward layers to preserve accuracy and attain efficiency of the model during pretraining and finetuning. A learnable pruning method implemented by

Yao et al. [104] to effectively prune the model based on thresholds learned by stochastic gradient descent. The method is different from other pruning methods because it utilizes a regularization function that connects with the preset target pruning ratio, unlike previous methods which use L0 or L1 penalty to indirectly affect the final pruning ratio. Another work Xu et al. [31] utilizes sparse pruning technique and knowledge distillation to present a method called Sparse- BERT, directly focused to BERT and executed at the fine-tuning stage. The observation drawn from the hybrid methods is that the combination were mostly made from structured pruning and unstructured pruning categories together with incorporating pruning with other methods of compressing pretrained language methods like Quantization, matrix factorization and Knowledge Distillation. We foresee more compression methods being combined with pruning and interesting results shall surface from hybridization. In addition, to determine the optimal layer size other proposed methods used regularization functions like group sparsity regularization and the L0, L1 penalty Alvarez and Salzmann [105].

2.3.2. Quantization

Quantization is one of the essential compression methods in the area of deep learning and is crucial for fine-tuning LLMs including BERT among others. The idea behide quantization is to reduce the memory footprint and computation costs of these models represent their parameters and activations using lower precision values, usually in fixed-point format or binary representations. This process not only simplifies the procedure of fast inference but allows the usage of models in different resource-limited devices like mobile phones and edge computing systems [106]. Quantization generally falls into two main categories: quantization-aware training (QAT) and post-training quantization (PTQ). QAT entails training the model by focusing on quantization such that the model is optimized for the quantization process at some training phase. This method calls for high computation power and involves an extra usage of the entire training data set; and thus, this is less parameter efficient [107]. On the other hand, PTQ is done after the training process, where it needs only a small subset of data for calibration. This approach is especially valuable when working with large models, because it makes it possible for deployment without the need for extensive retraining [106].

New trends in quantization are focused on dealing with challenges brought about by large models. For example, [106] uses dynamic per-token activation quantization and group-wise weight quantization; while the results are promising in maintaining accuracy while remarkably it can reduce model size [106]. The study by Zhang et al. [108] have argued that as models scale beyond a certain size, the effectiveness of traditional quantization methods can diminish due to the emergence of systematic outliers in activations. The idea has further proved by [106] indicating that it can lead to increased quantization errors. To counter these challenges, researchers have suggested the following: Outlier Suppression+ framework, which shifts and scales in order to suppress the outliers effectively Wei et al. [109]. This method enhances the quantization process by ensuring that the quantized representations can more accurately reflect the original continuous values, thereby reducing the accuracy degradation typically associated with quantization. The technique adds upon the quantization process by attempting to allow the quantized representation to be closer to the original continuous values, thereby minimizing the accuracy degradation normally observed during quantization. Moreover, methods like knowledge distillation have been applied in conjunction with quantization to enhance model performance while maintaining a compact size [108].

1. The Basic Concept of Quantization can be mathematically described as mapping a continuous range of values to a specific range of values. For instance, let a weight vector (W) in a neural network in which elements are denoted in floating points.

The quantization process defined as:

$$W_q = Q(W) = \text{clamp}\left(\frac{W}{S}, -2^{b-1}, 2^{b-1} - 1\right) \quad (9)$$

where:

(W_q) represent quantized weight,

(Q) represent the quantization function,

(S) represent the scaling factor,

(b) represent the number of bits used for quantization,

$(clamp)$ restricts the values to the specified range determined by b

The above formula illustrates how the continuous values of (W) are mapped to a discrete set defined by the bit-width (b).

2. In an 8-bit quantization scheme, weights are typically represented using 256 discrete levels (from -128 to 127 for signed integers), with maximum absolute value $\max(|W|) = 10$. The scaling factor (S) can be computed as:

$$S = \max(|W|) \tag{10}$$

For example, if the maximum absolute value in (W) is 10, the scaling factor would be:

$$[\, S \;=\; 10\,] \tag{11}$$

The quantized weight can then be calculated as:

$$W_q = \text{clamp}\left(\frac{W}{10}, -128{,}127\right) \tag{12}$$

The process reduces the model's memory usage, enabling more efficient computation.

3. Per-channel quantization permits different scaling factors for each output channel in a layer, which can lead to better performance. Mathematically, this can be expressed as:

$$W_{q,i} = \text{clamp}\left(\frac{W_i}{S_i}, -2^{b-1}, 2^{b-1} - 1\right) \tag{13}$$

where (S_i) denote the scaling factor for the (i)-th channel. This method is particularly useful in transformer architectures, where different channels may have different distributions of weights Bai et al. [110].

4. In quantization-aware training, the effects of quantization are modeled during the training process.. The loss function can be modified to include quantization effects, which can be represented as:

$$L = L_{\text{original}} + \lambda L_{\text{quantization}} \quad (14)$$

where $\left(L_{original} \right)$ is the original loss, $\left(L_{quantization} \right)$ is a term that penalizes deviations from the quantized values, and (λ) is hyperparameter that governs the balance between the two loss components Kim et al. [111].

5. Post-training quantization can be performed after the model has been trained. The process typically requires calibrating the quantization parameters using a small subset of the training dataset. The calibration can be mathematically expressed as:

$$S = \frac{\max(W) - \min(W)}{2^b - 1} \quad (15)$$

The above formula calculates the scaling factor determined by the highest and lowest values of the weights allowing for effective quantization without retraining the entire model [110].

Thus, as the need for efficient models increases, the current research in quantization techniques which is ongoing will help to form the advances of NLP and machine learning at large. It is expected that even more effective solutions for deploying large scale models in real world scenarios can be attained from the synergy of novel quantization approaches and other techniques that are aligned to them such as pruning and knowledge distillation as noted by Bibi et al. [112].

2.3.3. Knowledge Distillation

Knowledge distillation incorporates passing information or knowledge from a huge and complicated model called the teacher model to a simpler and more efficient one called the student model. The process objective is to preserve the performance of the teacher model while drastically decreasing the amount of computations needed in the inference phase.

Hinton et al. [113] introduced this concept, demonstrating that by aligning a student model with a teacher model through minimizing the difference between their output distributions, particularly using soft labels, effective knowledge transfer can be achieved. The primary motivation behind knowledge distillation is to overcome the challenges emerging through the integration of large models into real-world applications, which requires computation speed and minimal latency. For example, the studies by Zhao et al. [114] and Liu et al. [115] state BERT among others which possess millions of parameters can be resource hungry in terms of computation and memory demand. By employing knowledge distillation, researchers can create smaller models that maintain a high level of accuracy while being more well-suited for deployment on devices with limited resources, such as mobile phones and edge computing systems [6], [112].

Actually, there are different methods of how to apply knowledge distillation in practice. On of the most usual methods is to leverage the logits produced by the teacher model as soft labels in student model training. It then aids the student model to learn from the richer probability distributions generated by the teacher as opposed to relying solely on hard labels [106], Kim and Rush [116]. Furthermore, the latest developments have Integrated the teacher model's intermediate outputs to improve the effectiveness of student model training and provide better training transfer of knowledge as demonstrated by Yao et al. [117], Yao et al. [118].

Moreover, studies shows that it is possible to have a combination of knowledge distillation and other compression methods such quantization and pruning to enhance the student model's efficiency even further. For example, Q-BERT uses both knowledge distillation and quantization to yield ultralow precision quantization of BERT models as stated by Cui et al. [119], leading to substantial decreases in model size and inference time while maintaining high accuracy. This shows that knowledge distillation can be used in conjunction with other techniques in attempts to to optimize large language models. Since knowledge distillation is expected to improve with greater demand for efficient AI solutions, its importance is expected to grow in driving the development of future machine learning models Gu et al. [120]. Below

mathematical descriptions are provided of critical terms for knowledge distillation, especially in relation to LLMs.

1. The knowledge distillation loss function, as stated earlier, focuses on minimizing the difference between the predictions of the teacher and student models. Two components are said to be combined which are distillation loss and standard loss. This can be mathematically represented as:

$$L = \alpha \cdot L_{\mathrm{KD}} + (1 - \alpha) \cdot L_{\mathrm{CE}} \tag{16}$$

Where:

- (L) is the total loss.
- (L_{KD}) represent Kullback-Leibler (KL) divergence between the temperature-scaled probability distributions of the teacher and student models.
- (L_{CE}) is the cross-entropy loss with respect to the true labels.
- (α) is a weighting factor that balances the two loss components.

2. The purpose of applying temperature scaling to the softmax function is to smooth the teacher model's output probabilities, with the temperature parameter controlling this effect$T$ is introduced in the softmax function. The temperature-scaled softmax function is defined as follows:

$$p_i = \frac{e^{\frac{z_i}{T}}}{\sum_j e^{\frac{z_j}{T}}} \tag{17}$$

Where:

p_i denote the probability of class i

z_i are the logits (raw output scores) for class i.

T is the temperature parameter. A higher T results in a more uniform distribution.

3. Kullback-Leibler divergence is employed to assess how one probability distribution differs from a second, expected distribution. For two distributions P (teacher) and Q (student), the KL divergence is defined as:

$$D_{\mathrm{KL}}(P\|Q) = \sum_i \; p_i log \; \left(\frac{p_i}{q_i}\right) \tag{18}$$

where:

p_i is the probability of class i from the teacher model,

q_i is the probability of class i from the student model.

4. In Patient Knowledge Distillation (PKD) allows the student model to extract learning from various layers of the teacher model. The two methods can be mathematically represented as:

PKD-Last: The student extracts knowledge from the last k layers of the teacher model

PKD-Skip: The student extracts knowledge from every k -th layer of the teacher model

The loss function for PKD can be expressed as:

$$L_{\mathrm{PKD}} = \sum_{l \in \text{ layers}} \; L_{\mathrm{KD}}\big(f_l(x;\theta), g_l(x;\theta')\big) \tag{19}$$

where:

- f_l is the output of the l -th layer of the teacher model,
- g_l is the output of the l -th layer of the student model,
- θ and θ' are the parameters of the teacher and student models, respectively.

5. In the Uncertainty-Driven Knowledge Distillation is whereby an uncertainty estimation module (UEM) is introduced. The uncertainty of the feature distribution can be represented as:

$$U = \mathrm{Var}\big(f(x;\theta)\big) \tag{20}$$

where:

- U is the uncertainty estimate,
- $f(x;\theta)$ is the output of the teacher model for input x.

The modified loss function incorporating uncertainty is:

$$L_U = L_{\mathrm{KD}} + \lambda \cdot U \tag{21}$$

where:

- λ is a hyperparameter that controls the influence of uncertainty on the loss.

The mathematical explanations given above in this section provide a framework for understanding different ideas and techniques in knowledge distillation more especially in the context of large language models like BERT. Every part is important in enabling the student model effectively learns from the teacher model while maintaining performance and efficiency.

2.3.4. Neural Architecture Search (NAS)

Numerous studies have shown that NAS is a subdomain of ML which is more inclined to automation of neural network architecture designs. It has been found that its main goal is to discover architecture that improves performance on specific tasks while minimizing computational costs. Therefore, the idea of NAS is particularly relevant to LLMs like BERT, which have demonstrated significant capabilities in NLP but additionally, they present challenges because of their large size and high resource demand.

2.3.4.1. Search Space

Research indicates that the search space determines the range of architectures that can be evaluated during the NAS process. In the context of BERT, the search space may include a couple of variations in the number of layers, types of feed-forward networks configurations

and attention mechanisms. For example, search space definition can be a cell-based approach, where each cell represents a combination of operations (e.g. feed-forward networks, multi-head self-attention) that can be stacked to form the complete architecture Chitty-Venkata et al.[121], Pouy et al. [122].

Mathematical Representation:

$$A = \{A_1, A_2, \dots, A_n\} \tag{22}$$

where:

- A is the set of architectures,
- A_i denote i-th architecture,
- n represents the complete set of architectures within the search space.

A simple search space could be defined as:

$$A = \{\text{ BERT-base, BERT-large, DistilBERT }\} \tag{23}$$

This allows the NAS algorithm to explore different configurations of BERT architectures Xu et al. [123].

2.3.4.2. Search Strategy

Research indicates that the search strategy determines the method for navigating the search space. It can be framed as an optimization task aimed at identifying the architecture A^* that achieves the highest performance metric P.

Mathematical Representation:

$$A^* = \arg\max_{A \in A} P(A, \theta) \tag{24}$$

where:

- A^* is the optimal architecture,
- P represent a performance metric (e.g. latency, or memory usage).

Common strategies include: In this approach, Reinforcement Learning (RL) is a controller, often in the form of a RNN, they generates architectures which are then trained and evaluated. Some studies like that of Jaafra et al. [124] and Ren et al. [125] have considered that the performance of these architectures is used to update the controller's policy.

$\pi(A) =$ probability of selecting architecture A (25)

The performance feedback is used to adjust π to favor better-performing architectures., Klein et al. [126].

The Evolutionary Algorithms are said to simulate the process of natural selection, where a population of architectures evolved over generations based on their performance Bartz-Beielstein et al. [127].

The method of Differentiable Architecture Search (DARTS) is a technique that defines the architecture search problem using a differentiable approach, allowing for gradient-based optimization of architecture parameters as suggested by [126], Liu et al. [128]. Assessing the performance of candidate architectures is essential, due to traditional methods involves fully training each architecture, which is computationally expensive. Therefore some studies like that of Liu et al. [129], Klyuchnikov et al. [130] have demonstrated that methods like early stopping, weight sharing, and performance prediction can help reduce the computational burden. This strategy estimates the performance of candidate architecture without fully training them, which is computationally expensive. It can be represented as a function E that estimates the performance of architecture A.

Mathematical Representation: $E(A) \approx P(A, \theta)$

where:

- E is the estimation function,
- $P(A, \theta)$ is the actual performance metric of architecture AAA.

Early stopping or weight sharing techniques proves that they can be used to estimate performance. For instance, in weight-sharing NAS, by training a super-network, the performance of different architectures is approximated based on shared weights:

$$E(A_i) = \frac{1}{m}\sum_{j=1}^{m} P\left(A_i, \theta_j\right) \quad (26)$$

where:

- $E(A_i)$ represent estimated performance architectures A_i,
- m represent number of shared weights,
- θ_j are the shared weights for the architectures.

Chen [131] in their study suggest that the technique of AdaBERT uses differentiable NAS to compress BERT into smaller, task-adaptive models. The study further illustrates that it incorporates task-specific knowledge distillation and an efficiency-aware loss function to optimize the architecture for specific NLP tasks. A compression ratio of 11.5x to 17.0x was agained by AdaBERT model while maintaining competitive performance. The method NAS-BERT focuses on task-agnostic compression, allowing for the generation of multiple compressed models that is adaptable to multiple downstream applications. Studies shows indicated that search space includes feed-forward networks and multi-head self-attention where the training is performed on standard self-supervised tasks, making it adaptable to different applications [123].

The authors of a study which aimed at optimizing DistilBERT in a resource-constrained environment, inorder to explore a hierarchical search space an evolutionary algorithm was employed for that particular reason. The results on Yang et al. [132] suggested that efficient exploration could lead to significant improvements in performance while adhering to computational limits. Studies shows that an important aspect that is emerging from these

findings is that NAS can help researchers and practitioners to develop more efficient architecture that meet the demands of various applications through automation of the design process. It is evident from research that researchers continue to refine search strategies, expand search spaces, and improve performance estimation methods, paving the way for more accessible and effective use of LLMs in practical settings.

2.3.5. Low-Rank Factorization

According to the study by Pletenev et al. [133] low-rank factorization is reported as another powerful technique utilized in compressing large language models which are transformers based. The study further indicate that the main idea behind low-rank factorization is to approximate a large matrix, such as the weight matrices of neural networks, by breaking it down into the product of two or more smaller matrices. Several studies shows that this approach minimizes the parameter count in the model, which leads to less memory usage and computational requirements while attempting to maintain performance.

2.3.5.1. Matrix Decomposition

The idea behind matrix decomposition is to be expressed as like a matrix W of size $m \times n$ as the product of two matrices A and B, where A is of size $m \times r$ and B is of size $r \times n$. This relationship can be expressed as:

$$W \approx AB \tag{27}$$

Where r is the rank of the decomposition, which is less than or equal to the minimum of m and n. The parameter reduction is significant, as the total number of parameters in W is mn, while in the factorized form it is $mr + rn$. The work of Zhang et al. [134] mention that decomposing the weight matrix of a fully connected layer into two smaller matrices helps reduce the overall number of parameters while approximating the original output.

2.3.5.2. Singular Value Decomposition (SVD)

It has been reported that SVD is a specific method that is used for performing low-rank factorization. It factorizes the matrix W into three distinct matrices:

$$W = U \Sigma V^T \quad (28)$$

where:

- U comprises the left singular vectors,
- Σ represents a diagonal matrix containing the singular values,
- V comprises the right singular vectors.

To achieve a low-rank approximation, we can reduce U, Σ, and V by retaining only the top k singular values, yielding.

$$W_k = U_k \Sigma_k V_k^T \quad (29)$$

where U_k, Σ_k, and V_k correspond to the top k components.

In the paper by Ben and Zhu [135], SVD is applied to compress BERT, achieving a compression rate of 2x with a 5% drop in performance on the GLUE benchmark.

2.3.5.3. Fisher-Weighted SVD (FWSVD)

Detailed examinations shows that FWSVD improves traditional SVD by incorporating Fisher information to weigh the role of parameters in the factorization procedure. Fisher's information matrix F being utilised to adjust the singular value decomposition to prioritize parameters that significantly impact model performance:

$$W \approx U_k \Sigma_k V_k^T \quad (30)$$

where the weights in W are adjusted based on the Fisher information. FWSVD has been shown to maintain model performance better than standard SVD by focusing on the most critical parameters, allowing for higher compression rates without significant accuracy loss, Pletenev et al. [136].

2.3.5.4. Progressive Low-Rank Decomposition

Here in this method of low-rank decomposition is iteratively applied to the model during the fine-tuning stage. It can be described as:

1. Decompose the weight matrix W into two smaller matrices A and B.
2. Adjust the model through fine-tuning to restore accuracy
3. Repeat decomposition with smaller rank until the desired compression is achieved.

Hajimolahoseini et al. [137] mentioned that the progressive low-rank decomposition method allows for significant compression ratios which are greater than 8x while using knowledge distillation to improve performance.

2.3.5.5. Low-Rank and Sparse Approximation Approach

Low-Rank and Sparse Approximation approach like its name implies it combines low-rank approximation with sparse representation and the weight matrix W can be approximated as:

$$W \approx L + S \tag{31}$$

where L represent low-rank matrix while S represent sparse matrix. Decoupling allows for efficient compression while retaining important features Li et al. [138]. LoSparse method as discussed from their research effectively separates coherent and incoherent parts of neurons, allowing for better pruning of non-expressive components while maintaining model performance. The following methods like truncated SVD, FWSVD, and DRONE give valuable ways of optimizing model efficiency through parameter reduction while maintaining performance. As the demand for deploying AI models on edge devices continues to grow, the importance of these techniques will rise, driving further research and innovation in the field.

2.3.6. Weight Sharing and Clustering

In this subsection we discuss weight sharing and clustering as the two most important techniques used to compress large language models, optimize them for use in environments with limited resources.

2.3.6.1. Weight Sharing

Weight sharing is often described as the phenomenon where one uses the same weights for different sections of a neural network to minimize the total count of distinct parameters. This is especially helpful in large language models like BERT that can contain an extremely large number of parameters, reaching the scale of billions. It has been identified that the model can sustain overall performance while optimizing memory consumption through effective weight sharing. The process involves grouping similar weights and replacing them with a single shared value. If we have a weight matrix W of size $m \times n$, weight sharing can be represented as:

$$W' = \arg \min_{W' \sum_{i=1}^{k} \sum_{j \in C_i} \left|\left|W_j - c_i\right|\right|^2} \quad (32)$$

where:

- C_i represent weights set in a cluster i,
- c_i represent cluster's centroid.

For instance, in the ALBERT model by Lan et al. [139], its proposed modification is founded on the BERT architecture, the also authors employed weight sharing across all the encoder layers. It also enables the model to have a far reduced memory consumption yet achieves near optimal performance on different NLP tasks. In particular, ALBERT applies the same weights for all layers, that is, it has less parameters than BERT and, therefore, it is possible to train models that are larger and deeper without a direct increase in memory consumption.

2.3.6.2. Clustering

Clustering, especially weight Clustering means uniting close weight values into clusters and depict them through a single centroid or representative weight. It may be regarded as the method of quantization in which the main task is to minimize the bit representation of the weights, while the performance remains unaffected. When it comes to BERT, weight clustering can be performed on the embedding matrix or layer weights in the neural network. To give an example, when training a model, weights could be quantized to use K-means algorithm since weight values which are nearer to each other would be grouped together. Next, each group is represented by a centroid, this decreases quantity of distinct weights to be stored and retrieved during inference.

It is usually gained using algorithms like K-means, which partitions n observations into k clusters. K-means aims to minimize the WCSS (within-cluster sum of squares), defined as:

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{x \in C_i} \left| |x - \mu_i| \right|^2 \quad (33)$$

which:

- μ_i represent centroid of cluster C_i.

2.3.6.3. Combining Weight Sharing and Clustering

Weight sharing and clustering can be understood in two ways. steps of an optimization process. First, weights are grouped and can then be shared.

The combined objective can be expressed as:

$$\min W' \sum i = 1^k \sum j \in Ci \left| |Wj - ci| \right|^2 + \lambda \sum j = 1^m \left| |Wj - W'_{shared}| \right|^2 \quad (34)$$

where:

- $W'$is the shared weight value for the cluster,

- λ acts as a regularization factor that regulates the balance between clustering and sharing.

Formulae above allows for better compression through minimizing the total variance within the clusters while forcing weights similarity to similar clusters. Hu [140], performed a similar approach, where he used both a threshold pruning technique and KMeans++ weight sharing technique in attempt to compress the model to a compression ratio of 6.28x with the least loss of performance. This shows how these approaches can be employed together to achieve similar levels of accuracy with less overhead of larger models.

2.3.7. Parameter Sharing

The LLMs, such as BERT, use parameter sharing as a method of minimizing the number of parameters but still improve or possibly equal the models performance. This strategy is even more beneficial in cases where the resources available to run computations are scarce, for example, in mobile or edge implementation. Parameter sharing involves reusing the same parameters across different layers or components of a neural network. Within the BERT framework, parameter sharing can be implemented in several ways, including the following: In traditional models, each layer l has its own parameter W_l. In parameter sharing, this is simplified as: $W_l = W$ for all layers l

where a unified weight matrix$W$ is employed throughout all layers.

2.3.7.1. Layer Sharing

One approach is layer sharing, where identical weights are applied across multiple layers of the network. In the ALBERT model, cross-layer parameter sharing is implemented, meaning different transformer layers reuse the same parameters [139]. It effectively shrinks the model size and simultaneously facilitates the creation of more complex architectures [139]. This can be mathematically represented as:

$$h_l = f(W \cdot h_{l-1} + b) \quad \text{for all } l \tag{35}$$

where:

- h_l represents the output generated by the layer$l$,
- f represent the activation function,
- b represent the bias term.

2.3.7.2. Embedding Sharing

The approach of embedding sharing involves using the same embedding weights across tasks or layers, reducing redundancy. In models like TinyBERT, Jiao et al. [141] the embeddings are shared across different tasks, which helps in reducing the model size while retaining effectiveness. This can be expressed as:

$$E_{\text{shared}} = E_{\text{task1}} = E_{\text{task2}} = \cdots = E_{\text{taskN}} \tag{36}$$

where E_{shared} is the shared embedding matrix across tasks or layers, allowing for parameter efficiency without sacrificing performance.

2.3.7.3. Grouped Parameter Sharing

This technique of grouped parameter sharing it gives chances for sharing parameters among groups of layers rather than all layers, where each group has its shared parameters, but different groups may use different parameters. The process allows for some flexibility while still reducing parameter count across the network. The study by Liu at al. [142] in the FoldGPT model grouped parameter sharing is implemented where parameters from the corresponding layers of the parent block are shared with each child block. This can be mathematically represented as:

$$W_{\text{group1}} = W_{\text{group2}} = \cdots = W_{\text{groupN}} \tag{37}$$

Parameter sharing benefits are various, firstly it results in achievement of reduced model size which helps reduce the model and makes the model lighter to be deployed for running in environment which has limited resources. Second, it makes the training process more efficient, requiring fewer parameters to train, thus leading to the reduction of training

time and memory. Lastly, parameter sharing can either maintain or improve the performance, while in other circumstances, better generalization may be achieved as the model learns how to apply the shared parameter properly. Efficient large language models like BERT and its variants heavily rely on parameter sharing. Through decreasing the number of parameters, it makes such models more accessible for deployment in different environment regardless the available computational power.

2.3.8. Tensor Decomposition

Tensor decomposition is a mathematical technique that generalizes matrix factorization for use on higher-dimensional arrays, known as tensors. This method is sought to be useful in various domain such as machine learning, signal processing, data analysis etc., since it can simplify the representation of complex data structures. In the case of LLMs such as BERT, tensor decomposition can have a considerable impact on boosting model efficiency through model size reduction as well as improving computational performance without substantial loss in accuracy. A tensor, represented by a bold-faced letter, is a multi-dimensional array that builds upon the principles of scalars, vectors, and matrices. An n-order tensor can be represented as:

$$\mathcal{A} \in R^{I_{\mathbb{1}} \times I_{\mathbb{2}} \times \cdots \times I_n} \tag{38}$$

where $I_{\mathbb{1}} \times I_{\mathbb{2}} \times \cdots \times I_n$ are the dimensions of the tensor along each mode (axis). For example, a 3rd-order tensor $\mathcal{A}$ would be:

$$\mathcal{A} \in R^{I_{\mathbb{1}} \times I_{\mathbb{2}} \times I_{\mathbb{3}}} \tag{39}$$

As indicated by the study of Kolda & Bader [143] the notation indicates that $\mathcal{A}$ is a three-dimensional array with dimensions $I_{\mathbb{1}}$, $I_{\mathbb{2}}$, and $I_{\mathbb{3}}$ along its three modes. Below, we detail the fundamental concepts of tensor decomposition, providing mathematical explanations and examples for each.

2.3.8.1. Tucker decomposition

An advanced form of SVD in higher dimensions, Tucker decomposition represents a tensor using a core tensor and matrices applied across each mode [143]. For example, the tensor of a third-order can be represented by a core tensor and three factors matrices which will curtail the dimensionality of the parameters required to be stored for the tensor being decomposed. Tucker Decomposition can be considered as an extension of Canonical Polyadic Decomposition CPD; it breaks down a tensor into a core tensor and a set of matrices applied along each of its modes. In the case of a third-order tensor A, the Tucker decomposition can be expressed as:

$$\mathcal{A} \approx \mathcal{G} \times_1 \mathbf{U}^1 \times_2 \mathbf{U}^2 \times_3 \mathbf{U}^3 \tag{40}$$

where:

- $\mathcal{G}$ is the core tensor,
- $\times_n$ represents the mode$-n$ multiplication of a tensor with a matrix,
- $\mathbf{U}^1$, $\mathbf{U}^2$, and $\mathbf{U}^3$ are factor matrices corresponding to each mode.

In a multi-dimensional dataset, for an example user-item ratings in different categories Tucker decomposition allows us to identify all the hidden factors that could be influencing user preferences by storing all the information in a compact core tensor while preserving the relationships held by the factor matrices. In the context of BERT which take use of the transformer architecture, tensor decomposition can be applied to the weight matrices of the models. For example, in BERT, the fully connected layers can be expressed in the form of a tensor, these tensors can be approximated using a fewer parameter. This is especially advantageous because BERT has a massive number of parameters: it often ranges from hundreds of millions and leads to high memory usage and slow speed of inference [137].

2.3.8.2. Canonical Polyadic Decomposition (CPD)

Canonical Polyadic Decomposition (CPD), or PARAFAC, represents a tensor as the sum of rank-one tensors. For a third-order tensor $\mathcal{A}$, CPD is expressed as:

$$\mathcal{A} \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r \quad (41)$$

whereby:

- ∘ represent the outer product,
- a_r, $\boldsymbol{b_r}$ and $\boldsymbol{C_r}$ they represent the factors corresponding to each mode in the tensor,
- R is the rank of the decomposition.

2.3.8.3. Tensor Train Decomposition (TTD)

TTD represents a high-order tensor through a chain of smaller tensors, improving computational efficiency and storage. A tensor A in Tensor Train format can be expressed as:

$$\mathcal{A} \approx G_1 \times_1 \mathbf{X}_1 \times_2 \mathbf{X}_2 \times_3 \cdots \times_N \mathbf{X}_N \quad (42)$$

where:

- G_k are core tensors, each of which is lower-dimensional,
- $\mathbf{X}_k$ are factor matrices for each mode k.

TTD is mostly tailored for deep learning to provide compression large models by representing weight tensors in smaller version and using less memory and time for making computations.

When tensor decomposition is applied to BERT, it is often interesting to target the embedding layers as well as the feed-forward networks (FFNs). The embedding layer that takes tokens as input and returns dense vectors can be very large, especially with large vocabularies. Through tensor decomposition, the embedding matrix can be divided into a series of smaller tensors, while maintaining the same size as the original model and preserving its semantic depth [144]. For example, Liu et al. [145],-presented the work where it became clear that low-rank tensor decomposition of the embedding layers of BERT will allow achieving a decrease in the number of parameters by 8 times with acceptable accuracy degradation. It is accomplished on the basis of preserving the important features of the tensor in terms of

their significance and discarding others, thus making the model as effective as possible in its storage and computations.

Tensor-train decomposition applied to BERT is said to have retained or enhance the accuracy of the benchmark datasets such as GLUE while reducing the parameters by a very large margin [137]. Incorporating new techniques like gated multi-head attention allows the model to focus on more crucial features, making it possible to perform superior on various tasks like sentiment analysis and question answering Jin et al. [146]. Furthermore, to improve the model performance the following compression methods has been utilized to improve model performance after compression, tensor decomposition and Knowledge distillation. By using fewer parameters, researchers have reached results close to state-of-the-art performance [145], through the transfer of knowledge from a larger, pre-trained model to a smaller, compressed version. Through a reduction in parameters, the model accuracy is preserved, not only does tensor decomposition enable efficient deployment in resource-constrained settings, but it also improves the model's utility across multiple NLP tasks It remains clear that more work remains to be done in this field of research, which opens up the potential for additional progress in the field of model compression and efficiency.

2.3.9. Early Exit Mechanisms

Early exit strategies are applied in models like BERT to improve inference performance and speed by enabling the model to stop processing after intermediate layer rather than processing through all layers of the model. This approach is especially valuable when there is a constraint on computational power or where rapid responses are necessary, such as in real-time applications. This approach has mathematical fundamentals in confidence estimation, internal ensemble methods, and learned exits. Below are details of the fundamental concepts along with mathematical explanations and examples.

2.3.9.1. Confidence Estimation

This approach utilises entropy or on the maximum predicted class probability to estimate the confidence of the predictions occurred at intermediate layers. For example, if entropy of the predicted distribution at an internal classifier is less than some predefined value, it is possible to exit early [41]. Confidence estimation calculates a model's probability or, more formally, the degree of confidence regarding its outputs at each layer of a model, often quantified in entropy. For a given layer $\mathbf{L}_i$ with output probabilities $p(y|x)$ for class y given input x, the predicted probability distribution has an entropy H given by:

$$H(p) = -\sum_y p\,(y|x)\log\big(p(y|x)\big) \tag{43}$$

Model exits early if this entropy measure falls below a predefined threshold θ:

$$H(p) < \theta \tag{44}$$

DeeBERT incorporates entropy-based early exit where processing is terminated if the entropy of internal classifier's output distribution is low, implying high-confidence prediction Xin et al. [147].

2.3.9.2. Internal Ensemble Methods

This strategy utilizes multiple internal classifiers at multiple layers, and making exit decisions based on consistent predictions. For instance, the Patience-based Early Exit (PABEE) method involves exiting after classifiers a get to a particular number of consecutive classifiers to agree on a prediction before exiting [41]. The internal ensemble techniques use multiple classifiers within layers to evaluate the degree of agreed prediction, thus improving the robustness Zhou et al. [148]. This approach helps mitigate the risk of making premature exits based on unreliable predictions. For classifier C_i at layer i, the model can exit if:

$$\mathrm{Count}\big(C_i(x) = C_{i-1}(x)\big) \geq t \tag{45}$$

where t is the threshold specifying how many consecutive classifiers must agree for an exit to occur.

2.3.9.3. Learned Exits

Some methods, such as BERxiT, uses a learned mechanism to generate a score for the early prediction so that the model can exit as soon as the score exceeds certain threshold, He et al. [149]. Learned exits consist of training a scoring function to evaluate early predictions for accuracy. Let f be a function that generates a score s from an internal classifier C_i based on the hidden state h_i at layer i:

$$s = f\big(C_i(h_i)\big) \tag{46}$$

The model exits if the score s exceeds a threshold τ: $s > \tau$

2.3.9.4. Global Information Utilization

The function in global information utilization is to combine hidden states from all layers to make better prediction. The Global Past-Future Early Exit method augments past states along with the future states to provide more insights, as the previous methods only used the local data Liao et al. [150]. In BERT, the ability to compute early exit is achieved by adding internal classifiers at each of the transformer layer. In inference, if the model decides that it can make a confident prediction based on the output of an internal classifier, it can exit early, thus avoiding unnecessary computations in subsequent layers. This is highlighted by [148], where the authors use cross-layer consistency to develop a patience-based approach to decision of when to exit. This aggregation denoted as:

$$g(x) = \sum_{i=1}^{n} w_i \, h_i \tag{47}$$

whereby w_i are weights assigned to each hidden state h_i from the i-th layer. The model's decision is then based on this aggregated global information.

It is critical to design and place early exits with the network architecture to maximise their effectiveness. The research by Rahmath et al. [151] specifies that increased performance with lesser computational costs may be achieved if early exits are strategically placed at different layers. For example, there has been suggestions to use the Bag-of-Features (BoF)

models that assemble comprehensive yet lightweight representation from the feature maps at each exit layer in order to make better predictions with reasonable increase in complexity Passalis and Tefas [152]. In conclusion, incorporating early exit strategy mechanisms into the development of PLMs is a major improvement in improving the practicality of models to make them more efficient and adaptable to real-world applications. Using a confidence estimation strategy, internal ensemble methods, and hierarchical representations, researchers remain focused on enhancing early exiting approaches and applications for numerous natural language processing tasks [150].

2.4. Applications of Compression Techniques to BERT Models

2.4.1. Application of Quantization Techniques to BERT Models

The new methods of quantization have concentrated on PTQ, which enables quantization the pre-trained models without excess to retraining Xiao et al. [153]. Nevertheless, PTQ can result in large accuracy loss in the models with extreme outliers in the activation distributions Xiao et al. [154]. To overcome these problems, several techniques have been suggested such as Outlier Suppression+ framework in which to address outlier problem, channel-wise shifting, and scaling are used efficiently [109]. Furthermore, ZeroQuant has been also described as an efficient end-to-end quantization pipeline that merge fine-grained quantization schemes with layer-wise knowledge distillation provided ssubstantial compression of the model and faster inference with minimal impact on accuracy [118].

In addition, studies on integer only quantization techniques like I-BERT have shown that it is possible to perform inference entirely with integer arithmetic, which enhances the model support to hardware designed for low-precision computations Kim et al. [155]. Achieving competitive accuracy, this approach provides a very valuable advantage in terms of a computing speed, increasing the speed of inference, in comparative with the traditional floating-point methods, achieving up to a 4.0× speedup [155]. In summary, Applying

quantization techniques in deploying large language models is a promising approach to enhancing their real-world applicability and efficiency Zheng et al. [156].

2.4.2. Application of Knowledge Distillation

Although there are many other advancements in the area of knowledge distillation, by far the most famous one applied to BERT is DistilBERT, that reduces BERT to a smaller model but nonetheless remarkably portion of its performance. Here, DistilBERT employs both response-based and feature-based distillation where the student model is trained to mimic the predictions of the teacher model as well as the intermediate representations of the features. With a 60% smaller footprint, it delivers 97% of BERT's performance and operates 2.5 times faster Sanh et al. [157]. The Patient Knowledge Distillation (PKD) method improves conventional distillation by enabling the student model to leverage the teacher's hidden representations on multiple layers. PKD introduces two strategies: PKD-Last, PKD-Last involves knowledge transfer from the last (k) layers of the teacher, whereas PKD-Skip enables the student to learn from every (k)-th layer. This approach has shown superior performance and better generalization across various NLP tasks Sun et al. [158].

MobileBERT model is designed for resource-constrained devices and uses knowledge distillation to reduce BERT. MobileBERT is performed in a teacher-student protocol where the teacher is the larger BERT model pretrained on particular tasks, and a smaller BERT model acts as the student, learning to emulate the teacher model. This approach enables MobileBERT to retain high accuracy but also makes it scalable and compatible with the resource-constrained targeted mobile devices Sun et al. [159]. Another unique approach is Stochastic Knowledge Distillation (SKD) that also combines the problems of diversity loss and capacity gaps in the model compression methodology. According to the paper by Ding et al. [160], SKDBERT also reduces BERTBase by 40% with only an off 0.5% loss in language comprehension tasks, while also experiencing 100% boosts in inference speed [160]. This method stresses multiple teacher models and different distillation approaches to affect the student learning process. The current paper's exploration of PKD, SKD, and other knowledge distillation approaches

demonstrated their importance in ongoing work aimed at improving the usability of large language models for practical use in everyday life. When using these techniques, researchers can then gain the ability to reduce models such as BERT and still maintain their usefulness as natural language processing tools for use in constrained environments.

2.4.3. Application of NAS

The essential applications of NAS is the development of the task-agnostic compression strategies [123]. For example, the NAS-BERT framework uses NAS to shrink BERT models, but the shrunk models must be able to dynamically adjust to size and latency. This is highly relevant to deployment on devices with different memory and latency characteristics. NAS-BERT is based on the idea of training a large-scale model that integrates multiple potential architectures, in order for the architecture search space can be efficiently searched to identify find optimal configurations that are at the same time robust for downstream tasks. Another importance use is in the establishment of the adaptive-size BERT models. NAS-BERT not only can compress the model, but also the model size can also be adjusted according to the desired smallest size, according to the unique demands of the given task. This is made possible by spacing out the search space with the help of a technique that incorporates multi- head attention; separable convolutions as well as feed-forward networks, enabling the discovery of efficient architectures that meet different operational constraints [123].

As part of improving search efficiency in NAS frameworks, some strategies have been applied. For instance, block-wise search is used in the breakdown of the supernet into manageable blocks hence making the search space smaller significantly reducing the size G.M. and Pillai [161]. This method lets each block be trained by leveraging a pre-trained teacher model, which aids in preserving performance and at the same time minimizing computational costs. Moreover, performance approximation techniques are used to enhance the search process in order to make it feasible to conduct architecture searches even in rather constrained environments.

The paper on ‘’Optimizing DistilBERT’ clearly explained how one could implement an evolutionary algorithm within a limited GPU budget. This approach managed search space in a way that is computationally feasible, and demonstrated to democratize access to advanced NLP models. Since the study established the connection between pre-training and downstream tasks, pre-training validation was proposed as a criterion for model training Paraskeva et al. [162]. NAS techniques have also been used in defining the baseline for measuring the performances of BERT models. For example, NAS-Bench-NLP framework is a useful framework for designing benchmarks for Recurrent Neural Networks on textual data sets and enables both, intrinsic and extrinsic assessment of trained models. This benchmarking is important for evaluating the quality and the scope of discoverable NAS algorithms and their ability to address diverse NLP challenges and, thereby, enhancing the climate of the NAS and NLP research fields [130]. The utilization of Neural Architecture Search technique on BERT models is a major breakthrough in NLP field. NAS improves BERT model efficiency and scalability through automated architecture design, tackling issues related to resource constraints. This research area shows promising development in the near future if NAS is combined with BERT models, more innovative solutions can be derived for tackling various NLP problems.

2.4.4. Application of Low Rank Factorization

Among those several factorization strategies applied to BERT, a commonly employed technique is SVD. SVD breaks down a weight matrix into three matrices, so that dimensionality is reduced but it retains most relevant features of the weight matrix. For example, Noach and Goldberg [163] used SVD for compressing BERT by applying it to decompose every weight matrix in the model. They found that it can attain a 2x compression rate while only losing 5% of its performance on the GLUE benchmark Xu et al. [164]. This shows that SVD is useful in preserving the model accuracy in addition to minimizing the parameters needed as compared to the standard matrices.

Traditional SVD is not capable of weighing the significance of parameters during the factorization; that is why the Fisher-Weighted SVD (FWSVD) is a modification of the traditional SVD. Several authors have tried to use standard SVD, but they discovered its weaknesses, mainly, inability to take into consideration the level of importance of the different parameters; Hsu et al. [165] developed FWSVD to combat these drawbacks that were present in SVD. The study used FWSVD technique to compress BERT model with increased efficiency compared with the traditional SVD while the model performance remained unaffected. In their experiments, they demonstrated that FWSVD could compress the BERT models and the models achieved up to 30% compression ratio with minimal impact on task accuracy [165].

The analysis of low rank compression is done under DRONE criteria proposed by Chen et al. [166]. This approach acknowledge the fact that although BERT's weight matrices may not inherently be low-rank, the input vectors often belong to a subspace with lower dimensions. Overall, by focusing on the data distribution, DRONE's performance can increase its speed or compress data dramatically without requiring extensive fine-tuning. For instance, the DRONE model achieved 1.92x speed up enhancement on MRPC task that requires 1.5% reduction in test accuracy demonstrating its effectiveness in practical applications.

Another novel methods is Progressive Low-rank Decomposition which makes it possible to compress BERT models through iterative compression. The above idea was introduced by Hajimolahoseini et al. [137] in this approach, the transformer model's fully connected layers are broken down into smaller layers using a gradual SVD technique. This method enables compression during the fine tuning and is not as demanding as when pre training a new model from the ground up. In their respective tests, they postulated that it was possible to obtain better than 8X compression with these techniques providing reasonable performance.

LoSparse is a type of a structured compression method that utilizes a combination of low-rank and sparse techniques to compress large models, including BERT. This method can be used to achieve the low ranked structure while also preserving sparse structures which can

enhance the efficiency of the result. In another study, Li et al. [138] proved that LoSparse is capable of compressing BERT models with higher performance compared with conventional approaches and maintain accuracy across different tasks. For the BERT model, employing low-rank factorization method has also been demonstrated to have merits in model compression and efficiency. There are methods like SVD, FWSVD, DRONE, progressive low-rank decomposition, LoSparse that have shed light on how BERT's size can be decreased on par with its performance

2.4.5. Application of Weight Sharing and Clustering

Weight sharing is the process of categorizing similar weights in a neural network and putting all the individual values into a single shared value. This approach consequently eliminates the workload of having many unique parameters, which in turn makes the model small. For instance, the ALBERT model, a variant of BERT, employs weight sharing across its layers allowing it to maintain high performance while reducing memory consumption. In particular, all encoder layers of ALBERT have the same weights, and such an approach allows to train larger and deeper models without a proportional increase in memory usage [139]. The technique is particularly useful in those scenarios in which resources as memories and processing power are scarce, like in mobiles or edge computing.

It is also possible to apply clustering techniques, such as K-means clustering, for compressing the BERT models. In this context, weights are as a set of clusters and each cluster contains a centroid to represent all weights in the cluster. This method enables the model to achieve a considerable decrease in the number of weights that are unique. For example, Cho et al. [167] proposed the eDKM method in which K-means clustering is used on weight sharing during training to optimize weight sharing. Besides the compression of the model, this method also retains high accuracy by minimizing the weight variance, the model's capacity for representation is preserved. Hence, ideas on weight sharing and clustering are crucial in compressing BERT models in a way that expansions of their use can fit in numerous tasks, particularly in systems with limited resources. Through these mechanisms, the researchers

pave a way for use of large language models in real-time AI systems, mobile applications, and IoT devices. There is still ongoing research taking place in this field and focusing on improving the effectiveness and accessibility of NLP solutions.

2.4.6. Application of Parameter Sharing

One of the useful uses of parameter sharing in BERT is demonstrated in the ALBERT model where cross-layer parameter sharing is used. In ALBERT, all transformer layers use identical parameters, allowing for a deeper network architecture but generally makes the model lighter. This method enables the model to operate with a smaller parameter count as the base model of this proposal has only twelve million parameters, while BERT in contrast, has 110 million [139]. It also expends the shared weights spread across the layers which help in cutting down of redundancy and memory consumption which makes the model trainable on devices such as mobile phones.

Another example is seen in the TinyBERT model developed where knowledge distillation is used together with parameter sharing. TinyBERT applies shared parameters across layers, this proves helpful in size reduction but retains a massive portion of the BERT model performance. This model achieves a compression ratio of approximately 7.5 times smaller than BERT while preserving 98.5% of the performance on several GLUE benchmark tasks Jiao et al. [141]. Knowledge distillation and parameter sharing make it possible for TinyBERT to learn from the big BERT model while minimizing on the number of parameters required.

Besides, the authors propose a new parameter sharing mechanism that combines embeddings with hidden layers as part of the ShareBERT framework. This approach permits the compatible design of near-zero parameter encoders that work at 95.5% of BERT's performance using only 5 million parameters which is 21.9 times fewer than the original BERT model Hu et al. [168]. Parameters are shared between embeddings and the following couple of hidden layers in ShareBERT, proving that BERT can be compressed substantially, while

leaving knowledge distillation as a separate research direction, thus providing a new avenue for model compression.

In addition, to achieve efficient computation, FoldGPT model leverage something called grouped parameter sharing mechanism in which all blocks within a group utilize identical weights and additional trainable parameters. It also compresses the model while boosting performance on storage-constrained devices through higher cache hit rates [142]. The process of block removal and parameter sharing in FoldGPT also unlocks how parameter sharing is quite useful in attaining high levels of model reduction, especially while retaining the performance. Techniques for parameters sharing used in BERT models including the models described in ALBERT, TinyBERT, ShareBERT, and FoldGPT show how large amount of model reduction is possible without much effect on performance.

2.4.7. Application of Tensor Decomposition

One of the uses of tensor decomposition in BERT is reducing the size of the multi-head attention mechanism. The parameters or weights learned in each of the attention layers in BERT are in large numbers, which can be reduced using tensor decomposition methods such as Tucker decomposition and Block-Term Tensor Decomposition (BTD). As an example, Ma et al. [169] presented a new self-attention approach known as Multi-linear Attention where BTD is used to compress attention related representation. Specifically, it uses an efficient way to initialize a low-rank core tensor and employ the sharing factor matrices across the attention heads therefore greatly decrease the parameters yet increase or at least retain the language modeling performance.

Another important area where tensor decomposition applies to BERT for efficient compression of fully connected layers is Tensor-Train (TT) decomposition. Indeed, as far as the efficiency of the proposed approach is concerned, Xu et al. [170] showed, in particular, that TT decomposition can effectively reduce a large number of parameters in BERT models while maintaining its high accuracy. The TT decomposition enables high-dimensional weight matrices to be expressed as a sequence of lower-dimensional tensors, which can be processed

more efficiently. This method not only cuts down the memory complexity but also improves the inference time.

Similarly, techniques for tensor decomposition can also be incorporated with knowledge distillation toward improving the efficiency or BERT models. For example Pletenev et al. [136] looked onto the integration of Tensor Train Matrix (TTM) decompositions with Fisher information to align the objectives of the task and the decomposition process. This method, called FWTTM, showed that there was a significant enhancement in the proposed approach leverages tensor decomposition to facilitate parameter transfer from a large teacher transformer to a smaller student model, improving the performance of compressed transformer-based architectures.

Various low-rank tensor decomposition methods have been shown to enhance BERT model performance by eliminating redundancies in the weight matrices. Dantas et al. [144] also stressed that several of the embedding layers of BERT can be effectively reduced following the low-rank approximation, without a notable loss of the model's semantic information. This is especially ideal when the model is to be deployed on a restricted resources device as it allows for faster inference times without a substantial drop in accuracy. The further development of these methods is useful in helping to advance the development of better natural language processing solutions.

2.4.8. Application of Early Exit Mechanism

DeeBERT is one example where early exit in BERT has been utilized to state but one. This model has confidence-based early exit strategy, within which the model utilise entropy from the internal classifiers to exit early. If the entropy is lower than the predefined limit, the model proceeds with the prediction without processing every layer meaning that the inference time will be greatly reduced [147]. DeeBERT has been revealed to make the inference time reduce to 40% while maintaining minimal quality loss.

Another example such as PABEE (Patience-based Early Exit), which adds a patience mechanism into the process, where several internal classifiers to agree on a prediction before exiting. Besides, increasing efficiency, it also makes an algorithm more resistant to adversarial examples using the internal classifiers ensemble effect [148]. From this study, it can be concluded that PABEE is more accurate than traditional methods and this approach offers a more favourable balance between accuracy and computation time compared to other techniques that can be used in deploying BERT in areas of limited resources. The BERxiT framework applies early exiting to regression tasks and adds a learning-to-exit module to scores the correctness of early predictions. This allows the model to adaptively decide when to exit based on the complexity of the input, achieving significant speed-ups while maintaining high accuracy Xin et al. [171]. The flexibility in applying early exiting in regression tasks only further extends the use of BERT models for tasks across the finance sector, healthcare and others.

For sequence labeling tasks, the SENTEE and TOKEE mechanisms have been presented to extend early exiting for these applications. With SENTEE, a sequence of tokens exits simultaneously when uncertainty is minimal, while TOKEE permits partial token exits at varying layers based on localized information Li et al. [172]. These adaptations show the extendibility of early exit mechanisms for dealing with the difficulties of sequence labeling tasks, which involve token-level predictions.

In addition, by considering past and future contexts, the Global Past-Future Early Exit method improves the effectiveness of early exit strategies. This approach offers the benefit of enabling the utilize information from layers that have not been processed yet, leading to more informed predictions and improved performance [150]. Combining such information enables this method overcome challenges that the conventional early exit strategies present particularly because they use local representations. The early exit mechanism utilizing BERT models appear to be effective in enhancing the inference's speed for several task such as classification, regression, and sequence labelling. These techniques do not only offer computational savings but also improve or at least preserve model performance.

2.5. Iterative Magnitude Pruning (IMP) and the Lottery Ticket Hypothesis.

2.5.1 IMP

IMP is a method that has gained recognition as a clear and efficient way to achieve compression large neural networks. The concept used in IMP is to trim weights from a neural network depending on the magnitude of weights and subsequently, a fine-tuning step is performed to restore the accuracy lost due to weight pruning. This method reduces the parameter count in the model, resulting in notable gains in efficiency and faster inference.

2.5.1.1. The Process of IMP

The IMP process typically involves several key steps: model training, pruning weights, and fine-tuning the model. First, the neural network is trained until convergence, so the network can generalize the forms or patterns in the data. After training the model, the process of pruning begins. During this phase, specified number of weights with the minimum absolute values are set to zero, thus reducing the model. In determining which of the weights to prune, it is decided that those weights with the lowest absolute values are likely to be less influential in the computation of the overall output output [25].

Next is the fine turning phase in which a fine-tuning process is applied to the retained weights to recover the model's performance that is attributed to the pruning. Fine-tuning plays a crucial role in allowing the model to adapt to the more compact and sparse weight distribution. This pruning and fine-tuning can be iterated until the required level of sparsity is obtained. This kind of process assists in preventing degradation of model's accuracy while reducing the size of the model down Zhu & Gupta [34].

The significance of a weight w_i is gauged by its magnitude, $|w_i|$. For each iteration, the weights are pruned based on a certain sparsity level that has to be achieved. If the target sparsity is s, the smallest $s \times N$ weights, in which N represents the total number of weights, all of which are set to zero. The remaining weights are fine-tuned to help recover from the effect of pruning performed. The process proceeds iteratively until the required sparsity level

is achieved by the network. In [25] work, they employed IMP on the AlexNet architecture, which led to the removal 90% of the weight while maintaining the same level of performance, they argued that IMP is an effective way of achieving reduced model size with the least compromise on the model's performance.

IMP can be defined as an optimization problem aimed at minimizing the loss function $L(w)$ under sparsity constraints. Represented as:

$$\min L\,(w) \quad \text{subject to} \quad \left|\left|w\right|\right|_0 \leq k \tag{48}$$

In which $\left|\left|w\right|\right|_0$ is the l_0, denoting the count of non-zero weights, and k represents the desired number of weights to keep.

Weight Importance: The significancy of each weight w_i in a neural network is assessed based on its absolute value: $|w_i|$

Pruning Criterion: At each iteration of pruning, a set percentage p of weights with the least magnitudes are removed. Given that N represents the total number of weights, the quantity of weights to be pruned is: Number of weights to prune $= p \times N$

Pruned Weights: After pruning, weights are updated to zero based on a threshold t that reflects the desired sparsity level:

$$w_{\text{new}} = \begin{cases} 0 & \text{if} |w_i| < t \\ w_i & \text{otherwise} \end{cases} \tag{49}$$

Fine-tuning: After pruning, the model undergoes fine-tuning to minimize the loss function L, helping it to recover performance: $\min L(w)$

For a neural network with 1,000 weights, pruning 20% would mean removing the 200 weights with the smallest magnitudes. Afterward, the model undergoes fine-tuning to regain its performance.

2.5.1.2. Examples of IMP in Practice

An interesting work of Gordon et al. [86] based on the BERT model explores iterative magnitude pruning, and they proved that BERT's performance does not degrade, regardless of the presence of pruning subsequent to pre-training or fine-tuning. This result highlights the adaptability of IMP to different training schedules. Moreover, the authors found that a significant portion of the weights can be removed without risk such that the network's performance is significantly not compromised, a fact that is proof of the success of this method. Another example can be observed in the paper of [25] where the authors have introduced IMP in the work on deep compression. They showed that using IMP, the technique reduced weights in the AlexNet architecture by as much as 90%. This was achieved through the use of iterative pruning followed by fine-tuning, which helped the model effectively adapt to the reduced set of parameters.

2.5.1.3. Advantages and Limitations of IMP

A major benefit of the proposed technique is that IMP facilitates model sparsity by eliminating weights that are close to zero while retaining its good performance. It also suitable for deployment where memory and processing power are restricted, like in mobile devices or edge computing. Also, IMP is an iterative process that allows for a more controlled pruning process, thereby reducing the impacts of over-pruning which negatively impact the performance [34]. To begin with, it is essential to take the following aspects of IMP into account; It can be computationally expensive, as training and fine-tuning step are used repeatedly in the process. Therefore, this tends to impose higher training times particularly for large model sizes. Moreover, it is conceivable that while IMP works well for many architectures, it performs differently on the basis of the nature of the model and dataset [99]. IMP can achieve substantial reduction of model size while preserving parameter accuracy by repeated pruning of weights and fine-tuning of the model. Its application in various contexts, such as BERT and AlexNet, demonstrates its effectiveness and versatility.

2.5.2 The Lottery Ticket Hypothesis (LTH)

LTH first proposed by Frankle and Carbin [173] demonstrate that in a dense, randomly initialized neural network, several sub-networks, known as winning tickets, can be trained individually to perform on par with the original network. This hypothesis postulates that over-parametrization of neural networks is not simply a feature of deep learning, but a necessary aspect that enables the discovery of these winning tickets Chen et al. [174].

2.5.2.1. Understanding the Lottery Ticket Hypothesis

The LTH is predicated on the observation that neural networks often contain redundant parameters across the neural networks. According to the hypothesis, if such winning tickets can be identified and trained then what is sought to be suggested is that it is possible to achieve similar or even superior performance with a significantly smaller model. The process typically involves three main steps training a dense network, pruning it to identify the winning tickets, and then resetting the remaining weights to their initial values before fine-tuning Frankle et al. [175]. For instance, in their experiments, Frankle and Carbin have shown that one could obtain a sparse fully connected network with similar accuracy of one trained on the MNIST dataset through pruning. They discovered that these winning tickets could be found by iteratively pruning where weights with the smallest magnitudes were removed, and the retained weights were reinitialized to their original values [173], [174].

Mathematically, LTH can be described in terms of the loss function L of a network. A pruned model can attain equivalent loss performance L_{ticket} to original network's loss L_{full} after pruning and rewinding. The hypothesis suggests the existence of a sparsity pattern S such that:

$$L_{\text{ticket}}(S) \approx L_{\text{full}} \tag{50}$$

This implies that winning tickets retain the network's ability to generalize well.

LTH as an Optimization Problem: Identifying winning tickets in LTH can also be framed as finding a subnetwork $S \subseteq W$ that minimizes the loss while maximizing performance:

$$\min L_{\text{ticket}}(S) \quad \text{for} \quad S \subseteq W \tag{51}$$

in which W denotes all the weights present in the unpruned network.

Initial Network: Let θ_0 represent the initial weights of a dense neural network.

Masking: A binary mask m is created to indicate which weights to keep, defined as:

$$\theta_{\text{ticket}} = \theta_0 \odot m \tag{52}$$

where $\odot$ denotes element-wise multiplication. The mask m is created based on weight importance after training.

Pruning Criterion: The mask m selects weights based on their magnitudes in the trained model θ_T:

$$m_i = \begin{cases} 1 & \text{if } |w_i| \text{ is among the largest } k \text{ weights} \\ 0 & \text{otherwise} \end{cases} \tag{53}$$

Training the Ticket: The pruned subnetwork, or "winning ticket," is then trained to achieve performance similar to the original network:

$$L(\theta_{\text{ticket}}) \approx L(\theta_0) \tag{54}$$

Given a neural network with initial weights θ_0, after training, weights can be pruned down to 20% while maintaining performance. Applying the mask m to θ_0 creates the subnetwork θ_{ticket}, which is then trained to match the original performance.

2.5.2.2. Practical Applications and Examples

Surprisingly, for the LTH, their implications go beyond theoretical investigations; they were implemented in model compression and efficiency. For instance, [174] carried out LTH using pre-trained BERT models where subnetworks obtained during the masked language modeling task were later found to be transferable to other downstream tasks [31]. This

discovery shows that Smaller models derived from winning tickets can achieve similar efficiency and accuracy as their larger counterparts.

Furthermore, researchers have leveraged LTH in diverse areas, including NLP and computer vision. For instance, [23] conducted a study in which he checked the winning ticket properties of BERT models and concluded that even the least performing sub-networks could be further trained to give good performance; suggesting that a majority of weights in BERT contribute meaningfully to its performance. Such flexibility demonstrates the robustness of the LTH across significantly different architectures and tasks.

2.5.2.3. Challenges and Future Directions

However, besides its promising discoveries, the LTH has its limitations pointed out next. One problem is practicality — finding winning tickets entails training full models multiple times and is computationally expensive, especially for large models. Furthermore, studies shows that specific winning tickets can perform much better or worse depending on the initialization and it depends on the architecture that is employed [98], Cheong and Daniel [176]. Advancement in efficient algorithms for identification winning tickets and their application across various tasks and datasets may form the next research frontiers. The Lottery Ticket Hypothesis delivers a strong foundation for comprehending the possibilities of sparse subnetworks within over-parameterized models. Leveraging the existence of winning tickets, can enable future research to explore how to build more efficient models that possess high performance, for model compression and deployment in constrained contexts.

2.5.2.4. Relationship Between IMP and LTH

LTH marks the IMP as a practical strategy whereby winning tickets can be identified. IMP's iterative pruning and fine-tuning aligns with the LTH framework, where a subnetwork or winning ticket is identified through repeated pruning based on weight magnitude [174], [175]. The study by [23] prunes BERT using structured pruning which removes attention heads and multi-layer perceptron's (MLPs) layers. They found that even when pruned aggressively

down to their subnetworks, BERT could be retrained quite effectively, which supports the IMP hypothesis implying that during successive iterations these weights can be regained.

2.6 Attention head pruning in BERT.

Moura et al. [177] describe in their study how Attention head pruning in BERT decreases the quantity of attention heads within the multi-head attention layer. Researchers observed that different attention heads have varying impacts on accuracy, prompting them to leverage this finding for model optimization. The models can be compressed by deleting underperforming attention heads yet still maintain or enhance model accuracy. According to Qiu and Yang [178] BERT's multi-head attention mechanism uses multiple attention heads to attend to different portions of the input sequence at the same time. Studies suggest that distributing computations among multiple attention heads boosts the model's overall capability and efficiency. Several studies have found out that the reduced number of attention heads in models does not reduce performance quality. A study by [19] proves that keeping just 60% of BERT attention heads maintains accurate results throughout various applications. The analysis shows that there is redundancy in the attention heads, and that some heads may not be essential for the model's performance. Therefore, this indicates that not all attention heads are needed for accurate model output.

During attention head pruning we analyze the extent to which each head influences the model's predictions, which is the importance of each head. Several evaluation methods help measure head importance which include the change in loss function when a head is pruned or the average attention weights assigned to specific tokens during training. For example, pruning can be guided by the average attention weight that each head assigns to critical tokens, like the [CLS] token in BERT, which functions as a key element for classification tasks Rajapaksha and Crespi [179]. The model optimization reduces its size by finding and removing heads with low importance rankings.

Through attention head pruning you can boost inference speed and reduce memory usage with not requiring extensive retraining. The researchers Zhang et al. [180] created

Single-Shot Meta-Pruning to remove attention heads before fine-tuning, allowing for efficient adaptation to downstream tasks. The technique decreases model size and fastens fine-tuning to deliver better performance in systems with limited resources. We can use attention head pruning together with knowledge distillation methods to make the models smaller and improvements in efficiency. The Optimal BERT Surgeon (oBERT) system applies incorporates second-order that boost model accuracy while maintaining scalability [99]. Researchers can combine multiple model compression techniques to produce small versions of advanced systems which retain the performance of their larger counterparts, rendering them applicable across a wide range of NLP tasks. The process can be mathematically described and illustrated through various notations and examples as follows:

Multi-head Attention Mechanism: can be represented as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (55)$$

In which each head is calculated as:

$$\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad (56)$$

Here, Q, K, V correspond to the query, key, and value matrices, respectively, while W_i^Q, W_i^K, W_i^V denote the weight matrices that have been learned for each individual attention head i. Attention (QW_i^Q, KW_i^K, VW_i^V) calculates the attention for each head using these matrices, acquiring insights from separate representation subspaces for every head. The $\text{Concat}(\text{head}_1, \dots, \text{head}_h)$ represents the combined outputs from all h attention heads. W^O is the final transformation matrix applied to the concatenated attention heads.

Attention Function: defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (57)$$

In which d_k is the dimensionality of the keys.

Head Importance Score: The start of head pruning is measuring the significance of each attention head. The importance of each head shows through multiple assessment methods like averaging attention weight assigned to critical tokens (e.g., the [SEP] token) or the entropy of the attention distribution across different positions. The research from [35] showed that measuring how attentively the [SEP] token receives attention from each head helps identify candidates for removal. To determine which heads to prune, an importance score I_h for each head h can be calculated. One criterion could be the average attention weight to a specific token, such as the [SEP] token:

$$I_h = \frac{1}{N}\sum_{i=1}^{N} \text{Attention}_h\,(x_i, [\text{SEP}]) \tag{58}$$

In which N represent the number of tokens in the input sequence, and $\text{Attention}_h\,(x_i, [\text{SEP}])$ measures attention weight for head h focusing on [SEP].

Pruning Criterion: After calculating importance scores, heads can be removed from the model based on a predefined threshold. The network discards any heads whose measured values fall beneath a predefined standard. Michel et al. [19] performed research showing that removing a large portion of heads did not affect the model performance because extra heads provide redundant information. A threshold T is set to determine which heads to prune based on their importance scores:

$$\text{Prune}(h) = \begin{cases} 1 & \text{if } I_h < T \\ 0 & \text{otherwise} \end{cases} \tag{59}$$

where the function returns 1 if head h is pruned and 0 otherwise.

Fine-Tuning After Pruning: After pruning, the model requires fine-tuning to recover any performance that may have been lost. This step is important for the model because it needs this step to adapt its new structure and optimize the remaining heads for the specific downstream tasks. Fine-tuning is applied to the model to recover performance and reduce the loss function $L(w)$ while ensuring the pruned heads are excluded:

$$\min L(w) \quad \text{subject to Prune}(h) = 0 \tag{60}$$

where $L(w)$ is the model's loss function. Consider a BERT model consisting of 12 attention heads per layer. Post-training, the importance scores for each head are calculated by averaging the attention weight directed towards the [SEP] token. Suppose we obtain these scores: Head 1: 0.8, Head 2: 0.5, Head 3: 0.3, Head 4: 0.6, Head 5: 0.4, Head 6: 0.2, Head 7: 0.7, Head 8: 0.9, Head 9: 0.1, Head 10: 0.4, Head 11: 0.5, Head 12: 0.6.

When we use a pruning threshold T=0.4 our model will remove heads 3, 6, and 9 because these heads have lower importance scores than 0.4. The remaining heads will be kept by the model and undergo fine-tuning to recover any performance losses by adjusting the weights. It has been established through research that pruning attention heads leads to improved model efficiency. Michel et al. [19] found that removing up to 40% of BERT's attention heads did not impact its performance and even demonstrated the method ability to boost accuracy results. According to Voita et al. [35] their results revealed that reducing 38 of 48 transformer heads in an encoder system only caused a 0.15 decrease in BLEU score when translating from English to Russian on WMT dataset.

2.6.1. Attention head pruning effects on performance

Research shows that reducing many attention heads barely affects model efficirncy. According to the research by [180] they found that pruning 50% of the attention heads in BERT should not reduce its performance when executing tasks like machine translation or natural language inference. According to their results only a small number of heads carry unique information that helps the model to generalize well. Models respond differently to head pruning when completing distinct tasks. In machine translation research Rugina et al. [181], discovered that pruning half of the attention heads from encoder-decoder attention heads the approach decreased both latency and memory demands for attention-based models, supporting the creation of enhanced models for existing or new NLP applications. Conversely, self-attention heads could be pruned more aggressively without a noticeable impact on performance. Reducing attention heads improves both model size and speeds up

inference processing. BERT led to a speedup of approximately 17.5% in inference time for larger batch sizes [35], when half of its attention heads are trimmed. Systems that depend on real-time response can take advantage of increased performance through this enhancement.

2.6.2. Attention head pruning effects on Model Size

Each attention head in BERT increases the number of parameters in the model substantially. The study by [19] revealed that in BERT, around 8.34% of the model's total parameters are contributed by the multi-head attention mechanism. The total number of model parameters count can significantly drop when we remove these attention heads leading to a lighter model. The study by Behnke and Heafield [182] shows that a model size can be decreased at about 10% when pruning half of their attention heads. Fewer parameters result in a reduced memory footprint. The ability to maintain performance while reducing the model size makes attention head pruning an attractive option for practical applications [89].

2.7 Ensemble Learning Methods in Deep Neural Networks

Deep neural networks show better performance and robustness through ensemble learning as explained by Mohammed and Kora [183]. The strategy works well for deep learning projects since separate models often suffer from both overfitting and variance issues. Combining different models through ensemble methods allows the model to attain good accuracy and better generalization capabilities. According to Mienye and Sun [184] ensemble learning falls into three main categories: bagging, boosting, and stacking. Each of these methods employs different strategies to combine the predictions of multiple models.

2.7.1 Bagging (Bootstrap Aggregating)

Bootstrapping Aggregation (Bagging) an ensemble learning method aimed at enhancing machine learning models for more reliable and accurate outcomes while handling variance issues and overfitting challenges. The main goal of using bagging is to create several variant predictors and merge their output to develop a reliable model. Bagging uses multiple subsets of the training dataset Risch and Krestel [185] to reduce variance. Introduced by

Breiman [186], bagging involves creating multiple subsets of the training dataset through a process known as bootstrapping, where samples are drawn with replacement to lower prediction uncertainty. This method allows for the generation of diverse training sets, which are subsequently employed to train several models separately. The method combines the outcome of parallel trained models through common aggregation methods called averaging for regression tasks or majority voting for classification tasks to generate predictions [183].

The main objective is to boost predictive performance by leveraging diversity among the base learners. Bagging trains models on different subsets of data to reduce the likelihood that all models will make the same errors, thus improving the overall robustness of the ensemble Ganaie et al. [187]. Decision trees thrive with bagging because it controls their tendency towards overfitting. Ensemble learning methods create better predictions when combined because the aggregation process helps to smooth out the predictions, leading to a more generalized model that performs better on unseen data Khan et al. [186]. The research by Katpally and Bansal [188] shows that bagging methods make image classification neural networks more accurate by combining the predictions of multiple convolutional neural networks (CNNs) together on different subsets of training data. In addition, the study by Guo et al. [189] have demonstrated that bagging can effectively handle imbalanced datasets, where the distribution of classes is skewed, by guaranteeing that each model in the ensemble learns from a broad sample of the data.

Despite delivering benefits bagging faces practical restrictions. Training several models at once takes up high computer which may not be feasible for large datasets or real-time applications Cassales et al. [190]. Although bagging helps decrease reduces variance it does not solve model bias issues when individual base models perform poorly. This requires carefully balancing model complexity, computational efficiency, and predictive performance when applying bagging Altman and Krzywinski [191].

The bagging process involves the following steps:

Bootstrap Sampling: A large dataset of N samples produce many smaller bootstrap samples after random selection with duplicates. The algorithm creates multiple new samples by picking N random cases from the main dataset. As a result, some data points can appear more than once in one sample, while others may not appear at all.

Model Training: Separate basic models called weak learners are trained using random selections on each bootstrap sample. To generate M models require making M bootstrap samples from the source dataset.

Aggregation: The ultimate prediction is made by aggregating the predictions from all M model. When predicting results for regression tasks the system calculates an average of inputs and uses a majority voting approach for classification tasks.

Mathematically, the final prediction $\hat{y}$ in an ensemble is represented as:

$$\hat{y}_{\text{bag}} = \frac{1}{M}\sum_{m=1}^{M} h_m(x) \tag{61}$$

where:

- $\hat{y}_{\text{bag}}$ represents the final prediction, which is computed by averaging the outputs from all models in the ensemble.,
- M refers to the count of models in the ensemble, and
- $h_m(x)$ represents the output generated by the m-th model for the given input x.

This method decreases variation while improving prediction accuracy through aggregating multiple model predictions, especially when individual models produce high variance.

Bootstrap Sampling: From the original training dataset D of size N, M bootstrap samples D_m are created. Each sample is generated by randomly selecting N instances from D with replacement. The probability of selecting an instance x_i in a bootstrap sample is:

$$P(x_i \text{is selected}) = 1 - \left(1 - \frac{1}{N}\right)^N \approx 0.632 \tag{62}$$

This shows that approximately 63.2% of the original data points will appear in each bootstrap sample.

Model Training: A separate model h_m is trained on each bootstrap sample D_m. These model predictions are then aggregated for the final output.

Aggregation: The final prediction $\hat{y}$ can be expressed as:

For regression tasks:

$$\hat{y} = \frac{1}{M}\sum_{m=1}^{M} h_m\,(x) \tag{63}$$

Classification tasks:

$$\hat{y} = \text{mode}\{h_m(x)\}_{m=1}^{M} \tag{64}$$

In which M represent the number of models, and $h_m(x)$ denote the prediction from the m-th model. Bagging is theoretically grounded in the principles of bias-variance decomposition. The ensemble model's expected error is calculated as:

$$E[\hat{y}] = \text{Bias}^2 + \text{Variance} + \sigma^2 \tag{65}$$

where σ^2 is the irreducible error. Bagging primarily reduces variance by averaging multiple model predictions, leading to a more stable and accurate result.

2.7.1.1. Types of Bagging

Random Forest algorithm stands out as one of the top uses for the bagging technique. For Random Forest many decision trees are trained on random bootstrap samples and for each node split, a randomly selected subset of features is evaluated. By creating multiple random trees this method develops diverse results that boost the ensemble's performance [187]. Bagged Decision Trees is a straightforward application of bagging where each decision tree is trained on a unique dataset created through bootstrap sampling. For regression, the

method takes the mean of all tree predictions, while for classification, it selects the most common prediction. Yang et al. [192] states that bagged decision trees can reduce the variance as compared to a single decision tree. In bagging decision trees are the main base learners which are being used, even though other algorithms can be employed. Like for example, research shows that bagging works well with both support vector machines and neural networks where different models learn from various bootstrap samples, and their predictions are merged Chikhi et al. [193].

2.7.1.2. Examples of Bagging Application

Bagged decision trees provide better predictive results reducing variance than a single decision tree Zhou [194]. In their study Ali et al. [195] applied XGBoost gradient-boosting technology to find financial statement frauds. The model achieved 96.05% accuracy through SMOTE method handling of imbalanced classes while showing effectiveness of ensemble methods in real-world applications.

In bioinformatics scientists use bagging methods to make their models predicting gene expression and biological processes more reliable. The research paper [192] described how ensemble methods can improves bioinformatics analysis with effective handling of high dimensionality and small sample sizes typical in bioinformatics datasets. As reported by Zhang et al. [196] their analysis included examining decision strategies for ensembles particularly bagging in image classification tasks. The evaluation showed that unweighted averaging produced better results than alternative methods when each basic learner delivered similar test accuracies. Through multiple weaker learners over randomly selected subsets of data bagging improves model accuracy. Bagging solves the problems of data variance and helps generalize better models while maintaining its role as a necessary tool in machine learning applications.

2.7.2 Boosting

Boosting is commonly referred to as a prominent ensemble learning method boosting accuracy of machine learning models through combination of multiple weak learners to create a strong learner. Boosting uses an approach where models execute in sequence to confront and rectify past model errors. The model weights are assigned according to their individual performance levels whereby predictions are combined using a weighted sum. The method effectively minimizes bias while advancing ensemble predictive power. The theoretical base of Boosting rests upon the bias-variance decomposition principle. Extending from its core concept Boosting achieves lower error because it merges many weak learners together. The expected error of the ensemble is given by:

$$E[H(x)] = \text{Bias}^2 + \text{Variance} + \sigma^2 \tag{66}$$

where σ^2 is the irreducible error. Boosting reduces bias while maintaining a controlled variance. The mathematical formulation for boosting can be expressed as:

$$\hat{y}_{\text{boost}} = \sum_{m=1}^{M} \alpha_m \, h_m(x) \tag{67}$$

In which:

- $\hat{y}_{\text{boost}}$ represent final prediction coming from the boosting ensemble,
- M denote the number of models in the ensemble,
- $h_m(x)$ represent prediction from the m-th model for the input x,
- α_m represent the weight assigned to the m-th model, which is typically determined based on the model's accuracy or performance, Drucker et al. [197].

The systematic process of model weighting provides superior accuracy because boosting algorithms improve performance by focusing on challenging predictions through promoted model predictions. The boosting process can be summarized in the following steps:

Initialization: Begin with a dataset D of N instances. Assign equal weights to all instances, typically $w_i = \frac{1}{N} \text{for} i = 1,2, \dots, N$.

Model Training: For each iteration m (from 1 to M):

- Train a weak learner h_m using the weighted dataset.
- Compute the error ϵ_m for the weak learner on the training set:

$$\epsilon_m = \sum_{i=1}^{N} w_i \cdot I(h_m(x_i) \neq y_i) \quad (68)$$

where I it is the function that outputs 1 when the prediction is wrong and 0 when it is correct.

Model Weighting: Compute weight α_m of the weak learner based on its error:

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right) \quad (69)$$

This weight reflects the importance of the weak learner in the final model.

Update Weights: The instances will receive weight updates during the next iteration so misclassified instances receive increased weight and correctly classified instances get lowered weight values:

$$w_i \leftarrow w_i \cdot e^{-\alpha_m y_i h_m(x_i)} \quad (70)$$

Normalize the weights so that they sum to 1.

Final Prediction: The final model $H(x)$ this represents the sum of predictions from all weak learners, each weighted accordingly:

$$H(x) = \sum_{m=1}^{M} \alpha_m \, h_m(x) \quad (71)$$

2.7.2.1. Types of Boosting

a) AdaBoost (Adaptive Boosting)

The training strategy of AdaBoost updates instance weights after each weak learner run to highlight misclassified instances by giving them higher weights in the next iteration. A weak learner's incorrect classifications during binary classification steers instance weights to rise in future training cycles thus prompting the next weak learner to focus on them [197], Jovanovic [198]. The sentiment analysis task AdaBoost can utilize weak classifiers like decision trees to accomplish improved overall classification results [184]. Sequential weak classifier training occurs on AdaBoost while it reweights instances that get incorrectly classified by prior weak learners. The steps of the algorithm can be outlined as follows:

Initialization: (Set identical weights for every instance in the training dataset D:

$$w_i = \frac{1}{N}, \quad \text{for} i = 1,2, \dots, N \tag{72}$$

where N represent the number of instances.

Model Training: In each iteration m:

Train a weak learner h_m using the weighted dataset.

Calculate the error ϵ_m

$$\epsilon_m = \sum_{i=1}^{N} w_i \cdot I(h_m(x_i) \neq y_i) \tag{73}$$

In which I represent the indicator function.

Model Weighting: Determine the weight α_m associated with the weak learner:

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right) \tag{74}$$

Updating the Weights: Adjust the weights of the instances:

$$w_i \leftarrow w_i \cdot e^{-\alpha_m y_i h_m(x_i)} \tag{75}$$

Scale the weights to ensure their sum is equal to 1.

Final Prediction: The final model $H(x)$ represent weighted sum of the predictions:

$$H(x) = \sum_{m=1}^{M} \alpha_m \, h_m(x) \tag{76}$$

b) Gradient Boosting

Model building with Gradient Boosting occurs in stages that trains models to generate predictions concerning previous ensemble errors. The approach employs gradient descent to optimize a loss function during its operation. The rule for updating in the m-th iteration is expressed as:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x) \tag{77}$$

In which η represent the learning rate, and $h_m(x)$ denote the weak learner trained on the residuals. The core concept is to reduce a loss function L by progressively incorporating weak learners to address the mistakes of earlier learners:

$$F_m(x) = \operatorname{argmin}_F \sum_{i=1}^{N} L\left(y_i, F(x_i)\right) \tag{78}$$

The research work of Bentéjac et al. [199], showed XGBoost and other gradient boosting algorithms provided efficient multi-class classification while improving accuracy results. The optimized gradient boosting implementation called XGBoost uses regularization to prevent overfitting, used throughout machine learning competitions because of its strong efficiency along with scalability.

The objective function for XGBoost can be expressed as:

$$\mathcal{L} = \sum_{i=1}^{N} L\,(y_i, \hat{y}_i) + \sum_{j=1}^{K} \Omega\,(f_j) \tag{79}$$

where $\Omega(f_j)$ is a regularization term penalizing model complexity. XGBoost employs a second-order approximation of the loss function to achieve efficient optimization. The rule for updating in the m-th iteration is:

$$\hat{y}_i^{(m)} = \hat{y}_i^{(m-1)} + \eta \cdot h_m(x_i) \tag{80}$$

c) CatBoost (Categorical Boosting)

The CatBoost algorithm allows direct implementation of categorical features without the need for extensive preprocessing. Although based on gradient boosting principles CatBoost implements additional solution methods to handle categorical data efficiently. CatBoost transforms categorical features into numerical representations through methods which both maintain category-based information and minimizing model overfitting.

Categorical Feature Encoding: For a feature C with categories $c_1, c_2, \ldots, c_k$, the encoding for a category c_i computed as:

$$\text{Enc}(c_i) = \frac{\sum_{j \in \text{train}} y_j \cdot I(C_j = c_i)}{\sum_{j \in \text{train}} I(C_j = c_i)} + \tag{81}$$

where y_j is the target value for instance j, I is the indicator function, and ϵ is a small value added to avoid division by zero.

Gradient Boosting Framework: CatBoost follows a stage-wise process where each iteration m adds a tree T_m to the model. The cumulative prediction for an instance x after mmm iterations is:

$$F_m(x) = F_{m-1}(x) + \eta T_m(x) \tag{82}$$

In which η denote the learning rate.

The Loss Function: CatBoost minimizes a loss function L customized for the specific task. For binary classification, it can be expressed as:

$$L\big(y, F(x)\big) = -\Big[y\log\Big(\sigma\big(F(x)\big)\Big) + (1-y)\log\Big(1-\sigma\big(F(x)\big)\Big)\Big] \tag{83}$$

where σ represents the sigmoid function and y is the true label.

By providing controlled target information embedding, CatBoost helps prevent overfitting effects from traditional encoding approaches so it works best with datasets that contain numerous categorical features. A comparison between algorithms showed CatBoost delivered better performance than XGBoost and LightGBM in multi-class classification while maintaining superior generalization accuracy and AUC scores Odeh et al. [200]. CatBoost operates as a robust algorithm that excels in managing categorical features so it is useful across several applications, more importantly where categorical variables are prevalent.

2.7.3 Stacking

The use of stacking trains multiple models (base learners) followed by the application of a meta-learner for prediction combination Muslim et al. [201]. The individual base learners operate as diverse models yet the meta-learner trains using their prediction outputs. The mathematical representation for stacking is computed as:

$$\hat{y}_{\text{stack}} = g\big(h_1(x), h_2(x), \dots, h_M(x)\big) \tag{84}$$

where:

- $\hat{y}_{\text{stack}}$ represent the final prediction from the stacking ensemble,
- M is the number of base learners,
- $h_M(x)$ denotes the output from the m-th base learner for the given input x,
- g is the meta-learner, a model designed to aggregate the predictions of the base learners $h_1(x), h_2(x), \dots, h_M$ to make a more accurate final prediction.

Base learners training in multi-class classification problem can use the following algorithms together; CNNs, RNNs and support vector machine. Combining their predictions through a logistic regression model creates a meta-learner that produces higher accuracy Ghosh et al. [202]. A mathematical analysis of ensemble methods uses the bias-variance decomposition.

The expected error of an ensemble model can be expressed as:

$$E[\hat{y}] = \text{Bias}^2 + \text{Variance} + \sigma^2 \tag{85}$$

where:

- Bias^2 represents the **squared bias** of the estimator,
- Variance represents the **variance** of the estimator,
- σ^2 is the **irreducible error** (also known as the noise).

Base Learners: Let m be the number of base learners $h_1, h_2, \ldots, h_m$. Each base learner is trained on the dataset D and produces predictions $\hat{y}_i$ for each instance x:

$$\hat{y}_i^{(j)} = h_j(x_i), \quad \text{for} j = 1,2, \ldots, m \tag{86}$$

In which $\hat{y}_i^{(j)}$ represent the prediction of the j-th base learner of the i-th instance.

Meta-Learner: The outputs of the base learners serve as the input features for a meta-learner h_{meta}, which is trained on a new dataset D_{meta} that includes these predictions

$$D_{\text{meta}} = \{\left(\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \ldots, \hat{y}_i^{(m)}, y_i\right)\}_{i=1}^{N} \tag{87}$$

In which y_i represent the true label for the i-th instance.

Final Prediction: The final prediction $\hat{y}$ for a new instance x is obtained from the meta-learner:

$$\hat{y} = h_{\text{meta}}\left(\hat{y}^{(1)}, \hat{y}^{(2)}, \ldots, \hat{y}^{(m)}\right) \tag{88}$$

Bagging technique goal is to minimize variance while boosting operations minimize bias [192]. Stacking is based on both bias-variance trade-off theory to merge different model strengths while lowering error rates on prediction problems. The expected error of a stacked model can be expressed as:

$$E[\hat{y}] = \text{Bias}^2 + \text{Variance} + \sigma^2 \tag{89}$$

where σ is the irreducible error. By combining different models, stacking can achieve a lower overall error compared to individual models Van et al [203], Ju et al. [204].

Basic Stacking: Wolpert [205] conducted a traditional study by implementing stacking which involves combining trained base models and a meta-learner is used to aggregate their predictions. The final prediction are computed as follows:

$$\hat{y} = \sum_{j=1}^{m} a_j \, \hat{y}^{(j)} \tag{90}$$

where a_j are the weights acquired by the meta-learner.

The Super Learner: By applying cross-validation, the Super Learner improves stacking and optimizes the base learners' weights. The Super Learner optimizes the combination of predictions through cross-validated risk minimization according to Naimi and Balzer [206], Van Der Laan et al [207].

Cross-validated loss for the j-th base learner is computed as:

$$R_{\text{CV}}^{(j)} = \sum_{v=1}^{V} \sum_{i \in \text{val}(v)} l\left(y_i, p_{ji}^{-v}\right) \tag{91}$$

where p_{ji}^{-v} represent the prediction for the i-th is the instance from the j-th base learner trained excluding the v-th fold.

The robust method of ensemble learning in deep networks produces better model performance through collective features of multiple learners. Ensemble techniques including

bagging, boosting and stacking give different approaches to harness the advantages of each model while minimizing their limitations. Literature shows that ensemble methods' have the ability to generate outstanding results across image classification and natural language processing Jia et al. [208]. The ensemble methods maintain their essential role in deep learning research and application, driving advancements in predictive modelling throughout the evolving deep learning field.

2.8 Role and impact of subnetworks

Subnetworks in the context of model pruning which focused on transformer-based architectures such as BERT prove essential for systematic performance maintenance alongside improved model efficiency. Subnetworks represent smaller, pruned versions of larger models which maintain essential features and can be customized for specific applications. Research shows how combining multiple pruned models delivers both improved performance and efficiency.

2.8.1 Role of Subnetworks

The construction of subnetworks happens through pruning methods which eliminate key model components or less important parameters which include attention heads and layers. The model compression technique removes unimportant parameters while decreasing model size while eliminating superfluous network elements to minimize overfitting. Bian et al. [209] demonstrated how BERT models could benefit from pruning attention heads leading to size reductions without sacrificing performance levels because many attention heads proved redundant. Researchers can construct compact variants of language models using subnetworks that deliver similar performance results when applied to downstream tasks.

The implementation of diversified model types leads to enhanced ensemble method performance. The ensemble benefits from combining diverse subnetworks which separately process different architectures, of data structures or operate through different system designs to understand a broader array of patterns without overfitting. The study by [196] introduced

the fragmented neural network ensemble method, that involved training various weak neural networks with feature-sampled and sample-sampled data. This strategy permitted the ensemble method to extract beneficial capabilities from its subnetwork while decreasing the computational load involved in training a single large model.

The performance of subnetworks reaches peak effectiveness when subjected to fine-tuning methods. A study by [180], demonstrated that pruned subnetworks can undergo fine-tuning to regain performance on par with the initial model, demonstrating the adaptability of these smaller models. Because of their adaptability these models provide unique advantages they help achieve faster inference and decrease memory usage. The use of subnetworks produces substantial performance-based reductions in computational resource requirements across both training stages and inference phases. The research presents an approach demonstrating that training multiple smaller models, each with fewer parameters, is more effective than using a single large model, multiple subnetworks can be trained Sun et al. [210]. By parallelizing the training process the system both accelerates learning operations and facilitates distributed computing techniques.

2.8.2 Impact of Combining Multiple Pruned Models

Enhanced performance results can be achievable through the strategic combination of multiple pruned models leveraging the strength of diverse subnetworks. The integration of multiple pruned models provides better data representation since different subnetworks captures different points across the input data space. Held and Yang [211] demonstrated in multilingual model studies that combining subnetworks through pruning reduce interference and boost cross-lingual performance without the need for additional parameters tailored to each language. The integration of subnetworks results in a more reliable model which demonstrates improved generalization performance across different tasks.

The generalization abilities of pruned models surpass those of unpruned models because pruning minimizes overfitting effects. Multiple pruned models grouped together within an ensemble utilize the superior performance of each model component while

mitigating their individual weaknesses. A combination of multiple pruned models results in better prediction outcomes on new data sets. Research by Yang et al. [208] demonstrated that combining pruned deep learning algorithms together improved performance as compared to individual models throughout natural language processing operations. According to the study the ensemble delivered improved accuracy and robustness by seamlessly harmonizing the strengths of each pruned model.

When using pruned models together their combination results in reduced processing demands during both training and inference stages. By using smaller pruned models because they need fewer computational resources, the ensemble can operate efficiently. Real-time applications benefit most from this technique because it delivers essential computational efficiency. Research by Ju et al. [212] produced an ensemble of pruned convolutional neural networks which demonstrated equivalent accuracy to more extensive models yet used reduced time for inference. The combination exhibits optimal resource management for adequate deployment in networks with limited resources according to the authors.

Multiple pruned subnetworks can be integrated for producing a single superior model which minimizes performance reduction triggered by aggressive model pruning. A study in [34] showed that combining multiple pruned models resulted in better outcome performance compared to a single model because ensemble models distribute individual model errors among themselves. Better accuracy paired with stronger adaptability to diverse input data emerges when this method is implemented. This combination technique produces robust performance and enhanced results from multiple pruned models which proves beneficial for real-world transformer-based deployments. Modifying each subnetwork independently improves their operational outcomes while lowering computing requirements. These components ensure efficient machine learning development because they contribute to model diversity as well as generalization ability while cutting computational costs. These advanced techniques will strengthen their role in solving complex domain problems as the field matures throughout bioinformatics, natural language processing and computer vision domains.

2.9 Sustainability and Resource Efficiency in AI

Artificial intelligence technologies need improved sustainability along with resource efficiency because their widespread use across different industries grows rapidly. Focusing on environmental impact of AI systems while maximizing their operational efficiency. AI sustainability is the systematic creation and implementation of AI systems that produce minimum harm to the environment. Organizations should lower their energy requirements while they minimize carbon output and select sustainable renewable energy options. The concept of "green artificial intelligence" aims to solve the major energy usage problems which are associated with training and operating AI models. Alzoubi and Mishra [213] note that different organizations implement initiatives which focus on reducing environmental consequences through sustainability-based AI development practices. Google operates the AI Platform for Sustainability which incorporates cutting-edge AI solutions to tackle environmental issues. Through this platform businesses can leverage data-driven insights to achieve reduced environmental impact by optimizing their manufacturing energy consumption [213]. Through its workload optimization tools coupled with carbon footprint tracking the platform demonstrates how AI technology can serve sustainability objectives.

2.9.1 Key Initiatives and Tools for Sustainable AI

Optimizing cloud infrastructure stands as one of the main sustainability targets in AI while seeking to reduce power consumption. To solve environmental challenges through its platform Google AI Platform for Sustainability uses advanced AI technologies. Companies can leverage this system to make data-driven decisions that reduce environmental effects because it optimizes their manufacturing processes' energy usage Bolón-Canedo et al. [214]. This platform demonstrate how AI technology helps sustainability efforts through features which enable both workload optimization and carbon footprint measurement.

The sustainability of artificial intelligence requires enhancing the efficiency of AI models. NVIDIA Green GPU Platform and the Microsoft Azure ML Sustainability are used for decreasing the computing needs of AI models. The NVIDIA platform demonstrates

environmentally sustainable of deep learning operations by reducing power requirements through hardware optimization methods which deliver up to a 50% decrease Rashid and Kausik [215]. By optimizing performance these tools simultaneously contribute to reducing the environmental impact of AI operations. Assessment and optimization of AI systems carbon footprint stands as an essential factor for sustainability. The Climate Change AI Toolkit along with CodeCarbon deliver methods to measure AI workload carbon emissions. AI application monitoring tools provide organizations with environmental impact information about their systems and direct them toward sustainable choices [213]. Organizations can utilize these tools to detect regions of high energy consumption which helps establish operational sustainability goals. The EarthAI platform alongside Microsoft AI for Good Projects develop AI solutions that directly address environmental challenges. Through EarthAI's platform organizations can explore minerals without harming the environment as part of responsible resource management Rivas [216]. Microsoft AI for Good Projects use artificial intelligence to help minimize environmental sustainability problems in both deforestation detection and agricultural yield prediction, demonstrating the potential of AI to contribute positively to environmental sustainability.

2.9.2 Challenges and Future Directions

Multiple sustainability problems still hinder AI development despite the advancements. Large AI model training operates with excessive energy usage which generates substantial sustainability problems. The power consumption required to train models such as GPT-3 reaches 1,287 MWh because of which the process generates extensive carbon emissions according to reports [214]. Progress in sustainable AI depends on continuing research into better AI technologies which focuses on creating efficient algorithms while improving hardware utilization. The sustainability of AI features multiple complex approaches that are focused on reducing the ecological footprint of AI technologies. Cloud optimization tools alongside model efficiency strategies create essential pathways to sustainability in AI systems which ensure technology benefits are both environmentally sustainable. The research

presented in this section demonstrates how these topics will define AI technology development and its capacity to meet sustainability objectives going forward.

2.10. Resource Efficiency in AI

AI resource efficiency defines methods which optimize computational resource usage including energy, memory, and processing power, to achieve desired outcomes with minimal waste. The implementation of efficient algorithms alongside model optimization strategies becomes an effective method for achieving the goal of resource efficiency. The NVIDIA Green GPU Platform optimizes deep learning workload energy efficiency while achieving power consumption reductions exceeding 50% according to Waltersmann et al. [217]. This platform provides proof that combining hardware with software produces optimized resource management systems for AI engineering applications. The EarthAI initiative showcases its mineral exploration tools as they help reduce environmental damage. The EarthAI platform uses AI to develop resourceful methods which discover productive mining locations with improved environmental conditions for sustainable industrial practices [213].

2.10.1. Key Strategies for Resource Efficiency

Model optimization serves as a primary mechanism to improve resource efficiency rates. Methods such as model pruning, quantization, and knowledge distillation are frequently used together by researchers. Model size reduction with minimum accuracy reduction becomes possible through the process of weight pruning in a neural network according to Dantas et al. [144]. The hardware selection decision stands as a fundamental determinant of resource efficiency. The use of dedicated hardware like GPUs and TPUs leads to considerable enhancements in resource efficiency. The NVIDIA Green GPU Platform optimizes deep learning workloads by achieving an energy efficiency boost of 50% according to Shahid and Mushtaq [218]. This platform demonstrates how advances of hardware can contribute to more sustainable AI practices.

Resource efficiency stands as an integral part of contemporary cloud computing platforms. Through its use of advanced AI technologies, the Google AI Platform for

Sustainability helps businesses follow data-driven paths to reduce their environmental impact Jakkani [219]. Cloud services increase resource efficiency by offering tools which maximize AI application energy consumption. The concept of energy-aware deep learning applies optimization methods to accomplish lower resource consumption across training and inference operations. Reaching this goal involves implementing energy-efficient strategies, including dynamic voltage and frequency scaling, task scheduling, and power tracking [213]. Another study demonstrates that energy-aware deep learning techniques decreased energy utilization for large language models (LLMs) by 30% while retaining equivalent performance levels.

2.10.2. Challenges and Future Directions

Advancement in sustainable and resource-efficient artificial intelligence solutions face continuing difficulties. Academic research shows that the high amounts of energy consumption to train big AI systems creates major sustainability problems. The training of GPT-3 and similar models requires more than 1,287 MWh of electricity according to Tabbakh et al. [220] leading to carbon emissions. Research along with technological innovation must continue to address these issues.

Sustainable AI development requires three fundamental elements to flourish: the implementation of energy-efficient algorithms together with better hardware utilization and the deployment of regulatory standards that support environmentally friendly AI strategies. Research advancement in this field requires combined efforts of researchers alongside practitioners and policymakers according to Olawade et al. [221]. AI-based systems need resource efficiency development to create sustainable AI systems. The approach achieves optimal utilization of computational elements energy, memory, and processing power while maintaining or enhancing performance. The approach applies dual benefits because it enables both better environmental preservation of AI technologies and reduced operational costs.

2.11. Research Gaps

A detailed assessment of current methodologies shows how existing BERT pruning methods and ensemble subnetworks remain limited. The effectiveness of pruning techniques in reducing BERT model sizes contrasts with their difficulty in maintaining performance across various tasks. The widespread adoption of magnitude-based pruning has demonstrated ineffective performance when applied to transfer learning scenarios. Based on their research, Sanh et al. [6] demonstrate that magnitude pruning fails to capture the dynamic nature of weight importance during fine-tuning leading to poorer downstream task performance [10]. Contemporary pruning approaches demonstrate a fundamental weakness in their adaptive control of weight importance during the training procedure.

A fundamental gap exists because researchers have not systematically examined ensembles built from subnetworks that underwent pruning. The literature demonstrates how attention head pruning leads to efficient models with minimal performance loss [19], [35] despite lacking evidence for combining these pruned models as ensembles to improve generalization potential. Going forward the ensemble approach might solve performance problems that result from extreme pruning of subnetworks. The literature presents limited investigation into ensemble behaviours between multiple subnetworks which can leverage the strengths of each subnetwork to improve overall accuracy [35].

According to the lottery ticket hypothesis researchers have found compact subnetworks embedded inside large models which perform the same as the initial model [173]. The successful application of this hypothesis to BERT models remains an unexplored research topic. The work of [23] shows that truncated attention heads create valuable subnetworks but does not establish the stability of these subnetworks across different tasks or training runs. Researchers have yet to determine effective methods for discovering robust winning subnetworks in BERT-based systems.

Today's pruning methods fall short because they fail to implement coherent structured pruning approaches that permit full layers or attention heads rather than individual weights

removal. Studies show that structured pruning balances computational performance with substantial model size reductions [100]. Most existing research employs unstructured pruning methods yet real-world applications demand optimized hardware so these methods may prove ineffective. Standard hardware benefits from structured pruning for efficient inference yet unstructured pruning generates sparse networks that challenge hardware optimization [10].

Connections between pruned subnetworks for combined performance boost remain a major under-explored area in current research. The paper by [79] describes methods for developing compact models by pruning yet fails to show effective blending strategies that would merge individual subnets into a continuous ensemble system. Current analysis lacks critical research that demonstrates how ensemble approaches could enhance the generalization potential of pruned models. Research into transformer model attention head pruning has progressed substantially but additional study into how ensemble subnetworks should integrate with these pruning approaches remains inadequate. The future of efficient and robust natural language processing models could emerge from systematic evaluation of ensemble performance, dynamic importance of attention heads, integration of structured pruning, and the application of ensemble methods to the lottery ticket hypothesis. Future studies need to integrate these knowledge points to maximize ensemble technique effectiveness together with pruning approaches.

2.12. The need for advanced pruning framework.

Various critical factors associated with model efficiency, performance and deployment challenges motivate the development of advanced pruning frameworks within transformer-based models specifically for NLP. BERT as Transformer based model is commonly used in NLP, and their large parameter counts have brought the above challenges for practical applications, whereby it now calls for effective pruning strategies.

Firstly, transformer models, such as BERT, are characterized by their large parameter counts, which can exceed hundreds of millions. To illustrate, BERTBASE includes 110 million

parameters, in contrast to BERTLARGE with 340 million parameters [1]. This over-parameterization leads to significant computational and memory overhead, this makes these models unsuitable for deployment in environments with limited resources, like mobile devices or edge AI systems [89]. As highlighted in the literature, the demand for efficient models that achieving high accuracy with reduced size and latency is essential Kurtic et al. [222]. An advanced pruning framework can gradually decrease the model size without compromising performance, thus addressing these challenges. For example, the AE-BERT framework has demonstrated the ability to achieve a 75% pruning ratio on the QNLI dataset, delivering better accuracy than existing top-performing techniques.

Current pruning methods need users to do extensive hyperparameter tuning and manual intervention while requiring large amounts of time since these methods fail to achieve optimal outcomes [89]. For example, traditional methods often depends on heuristics to identify the to prune, leading to suboptimal pruning ratios and performance degradation Gupta et al. [223]. An automated pruning framework which works across different models and tasks can substantially boost both efficiency and performance according to research [179]. The Optimal BERT Surgeon (oBERT) framework surpasses current methods through its scalable second-order pruning strategies which deliver precise results [99]. The growing complexity of transformer models requires advanced pruning techniques because of their increasing requirements.

Existing research mainly analyse singular baseline models without studying how ensembles of pruned subnetworks might leverage the strengths of each subnetwork. The authors of [23] demonstrate effective subnetworks from attention head pruning yet they do not address stability between different training scenarios and tasks. An advanced pruning framework could facilitate the identification and utilization of these winning subnetworks, enhancing model robustness and generalization. Model generalization becomes more effective through proper pruning because the architecture's vital elements stand as the main focus of an optimized framework according to Li et al. [224].

The present research fails to acknowledge the valuable advantages that can emerge when whole layers along with attention heads are removed through structured pruning protocols [100]. Recent research indicates structured pruning methods can sustain computational efficiency while significantly reducing model sizes [100]. Research focuses predominantly on unstructured pruning methods which might not perform as well in practical scenarios where hardware efficiency is critical. Standard hardware typically performs better under structured pruning approaches yet unstructured pruning produces common sparse structures that make performance optimization challenging [61]. The advanced pruning framework is determined to maintain a balance model efficiency with performance, reduce the complexity of pruning processes, leverage insights from the lottery ticket hypothesis, and enable practical deployment of transformer models in diverse environments.

This chapter provided a comprehensive review of BERT's architecture, model compression techniques, and pruning methodologies, highlighting their significance in improving computational efficiency and sustainability. The discussion covered structured and unstructured pruning, the Lottery Ticket Hypothesis, attention head pruning, and ensemble learning, emphasizing their roles in optimizing transformer-based models. Despite these advancements, research gaps remain, particularly in the systematic integration of pruned subnetworks into ensembles and the development of automated, multi-objective pruning frameworks. These gaps underscore the need for innovative approaches that balance model performance, efficiency, and environmental impact.

Chapter 3 will present the methodology and theoretical framework of this study, detailing the proposed multi-objective optimization algorithm for BERT pruning and the Ensemble of Winning Tickets approach. The chapter will outline the research design, datasets (SST-2 and CoLA), preprocessing steps, and evaluation metrics (accuracy, F1-score, inference speed, and energy consumption). Additionally, it will describe the iterative pruning process and the validation techniques used to assess the effectiveness of the proposed framework. By the end of the chapter, the groundwork for experimental evaluation in Chapter 4 will be

established, demonstrating how the proposed methods address the limitations identified in the literature.

Chapter 3: Methodology

This thesis utilizes a positivist paradigm in its methodology to pursue empirical research, rigorous experimentation, and quantifiable results. Ali [225] shows that positivism centres itself on the comprehension that reality remains objective because scientists can measure it through standardized empirical approaches. Within this research framework scientists create research results that maintain scientific integrity for replication and offer generalized conclusions. BERT's computational and environmental challenges demand the positivist paradigm to validate whether the proposed advanced pruning approaches show effectiveness. Experimental data collection and analysis supports this research as it measures the proposed methods against performance metrics including as model accuracy, F1 score, computational cost, inference speed, and energy consumption.

3.1 Research Paradigm

As a fundamental research paradigm the positivist framework puts physical evidence alongside scientific research methods at its core. Research into BERT large-scale language model applications uses this paradigm to allow systematic evaluation of compression techniques through experimental validation. According to Auguste Comte's original construct key members of the Vienna Circle expanded his theory which holds that scientists should exclusively use empirical data while studying observable events Holden and Lynch [226]. The methodology operates from the basis that researchers can measure and analyze objective realities by conducting stringent experimental investigations. The study of model compression techniques can benefits from statistical methods which enable assessment of BERT model performance after pruning.

Research has extensively studied the model compression technique known as pruning which removes neural network weights or neurons to minimize network size while maintaining inference speed and accuracy levels [7], [25]. Researchers using positivist methodology can provide measurements of pruning impacts by capturing key metrics

including accuracy, F1 score, and inference speed. before and after the application of pruning techniques. Weight pruning methods were tested by Gordon et al. [86] showing how these methods could compress BERT without sacrificing performance in downstream tasks.

Under a positivist framework research requires reproducible results that can apply to broader populations. Research findings regarding model compression effects in natural language processing (NLP) gain broader acceptance through the application of established evaluation frameworks such as GLUE and SQuAD [11]. Systematic methodology improves both research validity and also facilitates the development of best practices for implementing model compression techniques in real-world applications. By using a positivist paradigm to investigate model compression methods for BERT and similar large-scale language models researchers can systematically analyse data-driven evidence with scientific procedures. Through observable phenomena analysis alongside quantitative measurement the positivist approach enables a sophisticated understanding of model compression operations in natural language processing settings.

3.2. Research Design

The experimental framework of this research is structured to systematically investigate and optimize pruning techniques for BERT models, with a focus on enhancing computational efficiency, maintaining task performance, and improving energy sustainability. The design integrates three major contributions: a conceptual framework for pruning BERT attention heads, a multi-objective optimization algorithm, and the creation of ensembles of neural subnetworks/ winning tickets. These contributions collectively address the challenges of balancing model size reduction with computational and performance trade-offs.

3.2.1. Problem Definition and Research Objectives

This PhD thesis address efficiency improvements for BERT through attention head pruning approaches designed to boost performance while minimizing computational overhead and energy consumption. The first part of the study develops a thorough framework for attention head pruning which identifies less significant components in the attention

mechanism and systematically remove them. Simultaneously, a multi-objective optimization algorithm is applied to find an optimal balance between task accuracy, inference time, and energy consumption. Additionally, we explore the creation of neural subnetworks (winning tickets) that can independently perform well on tasks and be combined into ensembles for increased robustness and adaptability.

3.3. Proposed Pruning Framework

This research presents the new sparsification-driven framework in an effort to obtain pruned models for sustainable technological development. The proposed framework aims at exploiting the idea of sparsity in deep learning to map pruned models into resource limited devices in line with long-term technology roadmap. Efficient models are those that refine metrics such as model size, latency during inference, and time required for training while minimizing quality loss and improving model generalization. Sparsity in deep learning enables compression techniques to target the representational efficiency of over-parameterized models Menghanin [227]. “Sustainable technological development involves articulating functions to meet future demands and societal needs while reducing environmental impact” Vergragt & Jansen [228] .

The sparse centered to sustainable technological development approaches are currently lacking. The framework aims to make BERT and other large language model more efficient, sustainable, and effective in real-world applications by demonstrating techniques for reducing their size, optimizing their training schedules, and considering practical deployment factors. Thus, the research that presents a conceptual framework based on the analysis of sparsity in scientific publications will help to contribute to achieving goals of a sustainable technology development for creating environmentally friendly and sustainable technologies.

The following steps were taken to develop the framework:

- An analysis of literature on the fundamental ideas pertaining to the key research topics that include model pruning, efficiency, inference, and sustainable information technology.
- identifying connections between research concepts.
- design and development of the proposed conceptual framework for pruning.

3.3.1. Reviewing Literature on Specific Research Concepts

This study has undertaken a rigorous and structured approach to determine the relationships between the key concepts of this research. The structured approach is based on Kitchenham guidelines Kitchenham & Brereton [229], which include the preparation of research questions, a search strategy, and research selection criteria.

3.3.2. Planning the Review

The general research question which was identified is as follows: How can model pruning be used to enhance efficiency and support sustainable technological development in deep learning models? To make it possible to further explore the problem, the general question was divided into three narrower sub questions.

1. In what way does pruning help make large language models more efficient?
2. What is the association between model efficiency and inference performance in sparse models?
3. How exactly does increased model efficiency impact the further development of sustainable technologies?

3.3.3. Developing a Review Protocol

The aim is to establish a conceptual framework that supports the implementation of pruning methods for improving the efficiency and sustainability of deep language models. The search strategy in this study, the key concepts represented the search terms that were applied to a variety of resources. Boolean search strategy is defined as the process of converting a

research question into a research string with the view of querying a database or search engine, Sivarajkumar et al. [230]. The academic internal databases employed in this study include semantic scholar, Google Scholar, PubMed, IEEE Xplore and ACM Digital Library. Keywords are "model pruning," "deep learning efficiency," "neural network optimization," and "sustainable AI.

To fulfill the inclusion criteria, we restricted our sources to the articles from the referred scientific journals, evaluation reports and technical papers and conference papers of peer review. More precisely, we only considered papers that focused on the effects of pruning on the model as well as papers that focused on the broader issue of sustainability in AI and deep learning. We prioritized articles that demonstrated a link between the search terms and described the relationships between them. Additionally, we included articles addressing the search terms in conjunction with any of the four core concepts of this study: model pruning, efficiency, inference, and sustainable technology development Shao & Zhang [231].

The articles were excluded based upon the following: articles that did not relate directly to deep learning or pruning techniques, papers and articles with no prior experimental results or data, the papers with the search terms but not looking into the connection or relevance between them were considered irrelevant and thereby excluded. To achieve the data extraction methods, we created a standardized data extraction form to capture relevant information from each study, including pruning methods, metrics for efficiency, impacts on inference, and sustainability considerations.

3.3.4. Conducting the Review

The identification of research was achieved by using the ‘AND’ and ‘OR’ Boolean operators to construct search strings to extract articles that join terms based on the research questions. The search strings included the following "model pruning AND deep learning efficiency," "pruning techniques AND inference performance," and "sustainable AI AND neural networks." The strings for the OR operator included "model pruning OR deep learning

efficiency," "pruning techniques OR inference performance," and "sustainable AI OR neural networks."

In the process of selecting primary studies, initially, we searched and selected studies based on the titles and the abstracts of the studies and then, we checked and selected the full text of these studies. Of 500 original identified articles, 120 papers were reviewed after excluding irrelevant and low-quality articles while 46 were reviewed in this final review. Data was systematically extracted using the predefined form and the extraction process was monitored to maintain consistency and accuracy. Some of the data considered during extraction involved the type of pruning technique applied whether structured or unstructured, efficiency measurements such as model size and inference latency, and whether sustainability impacts have been realized such as energy efficient consumption.

3.3.5. Identification of relationships between key concepts

These relationships were formed through an extensive examination of relevant studies in the field. To find studies that investigated how pruning influences the performance of neural network models, we concentrated on peer-reviewed journal articles, conference papers and technical papers that described certain approaches to pruning and their results. Also, the relevant literature concerning sustainability in AI and deep learning was discussed. Any articles that showed utilization of the search terms and explained the correlation between the identified search terms were included into the study. This involved reviewing different articles to see how model pruning affects efficiency and inference and how such gains fitted sustainable technology progress Singh & Gill [232]. We began by analyzing the effect of model pruning in model efficiency. For instance, Tay et al. [233] and Cho et al. [234] showed that pruning led to smaller models.

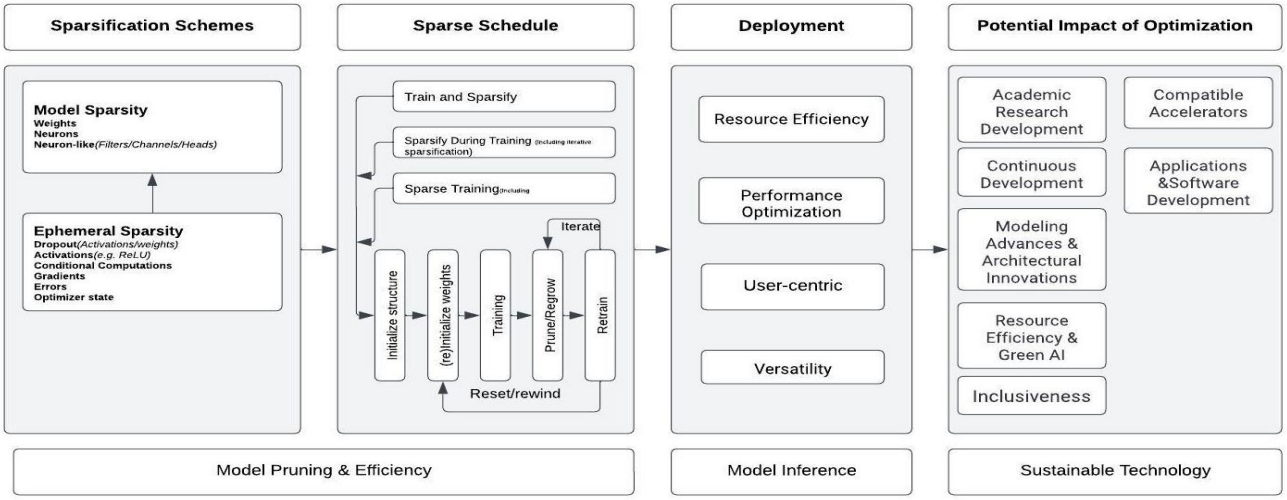


Figure 3: Proposed Conceptual Framework for Pruning of neural networks

3.4 Multi-objective Optimization Algorithm for Performance, Speed, and Resource Efficiency.

We propose efficient model pruning optimization algorithms that are crucial for reducing the size and computational demands of large language models, particularly large-scale transformer-based architectures like BERT. The algorithm has been designed to detect minimal parameters which do not affect model performance thus enabling deployment across resource-limited systems that include mobile devices and edge-based AI systems.

3.4.1. Efficient Model Pruning Optimization Algorithms

Researchers have developed various algorithms to improve model pruning methods by balancing model size reduction with performance retention. Here are some notable approaches: Through the Iterative Magnitude Pruning (IMP) technique, the model is pruned iteratively, with retraining steps aimed at restoring any reduced accuracy. The approach first trains models to achieve convergence before using magnitude-based pruning to eliminate weights of lower magnitude. The model returns to its initial weight configuration after resetting all remaining weights [173]. This methodology successfully picks out "winning tickets" which represent subnetworks that deliver equivalent performance to the original model according to research by [37]. Movement Pruning method which is proposed by [6],

the technique Emphasizes preserving weights that show significant deviation from zero during fine-tuning, rather than simply preserving those that are far from zero. Researchers have demonstrated this approach produces better results when trained for transfer learning because the pruning method tracks the model's training behaviour [7].

The Differentiable Pruning method allows pruning decisions to be made dynamically during training. The model incorporates pruning as a differentiable process, allowing it to learn which weights are essential during training. This method can lead to more optimal pruning decisions without the need for separate retraining phases [68]. The weight importance guide from Upper Confidence Bound (UCB) Pruning by Zhang et al. [235] this approach utilizes the upper confidence bound of weight relevance to inform pruning decisions. The technique enables assessments of uncertainty impacts on weight contributions to model performance, so that practitioners can make better pruning choices.

Through our Efficient Model Pruning Optimization Algorithm we have developed a structured approach to deep learning model pruning which optimizes both performance and cost while controlling both inference time requirements and power usage. Before the method commences there is a requirement to establish initial weights W together with model accuracy measure $A(W)$, acceptable accuracy loss threshold ϵ and computational cost $C(W)$, real-time execution $T(W)$, energy utilization $E(W)$ and optional pruning ratio p. A target pruning ratio p directs the model pruning process which identifies the smallest weights based on their magnitude to remove $p \cdot 100\%$ weighted elements from weights W. The computation evaluation $C(W')$ of the pruned model follows to confirm significant computational savings with maintained acceptable accuracy levels $A(W')$, where $A(W') \geq A(W) - \epsilon$.

The system assesses inference efficiency by measuring time $T(W')$ to perform inferences while achieving desired accuracy levels. The sustainability evaluation of the pruned model happens through an examination of its energy consumption $E(W')$ which requires optimization under identical accuracy requirements. When initial pruning leads to suboptimal output the algorithm dynamically adjusts parameter p to align efficiency with performance

needs. A combined objective optimization run minimizes W\prime's computational cost $C(W')$, $T(W')$, and energy utilization $E(W')$ while maintaining accuracy levels higher than predefined standards. Multi-objective optimization technique helps users identify optimal trade-off points. After meeting the objectives, the proposed algorithm produces both W' and its performance metrics. The method provides a structured way to perform model compression by balancing precision and speed alongside sustainability performance.

The method builds a system for reducing deep learning models through systematic approaches which enhances their performance combined with lower computational demands and decreased inference time requirements as well as increased resource efficiency. Below is a detailed explanation of each step, with examples for better clarity:

Input Definitions

1. **Original model weights** $(\boldsymbol{W})$**:** The parameters of this model exist as $\boldsymbol{W}$ in an unpruned state commonly used for a pre-trained BERT model.
2. **Accuracy of the original model** $\boldsymbol{A(W)}$**:** The task performance metric of the original model, e.g., accuracy on a sentiment analysis task.
3. **Acceptable accuracy loss threshold** $\boldsymbol{\varepsilon}$**:** Pruning operations need to maintain accuracy performance within an acceptable range of ε percent from baseline performance. A $\varepsilon = 1\%$ threshold applied to the 90% baseline accuracy standard requires the pruned model to maintain higher than 89% accuracy.
4. **Original computational cost** $\boldsymbol{C(W)}$**:** Models normally quantify their computational costs through FLOPs (floating-point operations per second) measurements.
5. **Original inference time** $\boldsymbol{T(W)}$**:** The model process time duration as measured during input processing while executing in milliseconds.
6. **Original energy consumption** $\boldsymbol{E(W)}$**:** Reality-based inference needs energy measured in Joules.
7. **Target pruning ratio** $\boldsymbol{p}$**:** A value that defines the fraction of weights for pruning when this amount remains the target or when dynamic adjustments occur

Step 1: Model Pruning

1. **Initialization:** The model reduction process begins with all of its complete weight parameters W.
    - *Example:* A BERT-base model with 110M parameters.
2. **Set target pruning ratio p:** Choose which fraction of weights to remove from the model space for values between $0 \leq p \leq 1$.
    - *Example:* The target pruning scheme chooses to remove a ratio $p = 0.3$, which means removing 30% of weights from the initial network.
3. **Prune weights:** Choose all weights with smallest magnitude from the pool according to the target pruning ratio $p \cdot 100\%$ for elimination.
    - *Example:* If $W = [0.8, -0.5, 0.03, -0.1]$ and $p = 0.5$, prune the smallest two weights $(0.03, -0.1)$, resulting in $W' = [0.8, -0.5]$.

Step 2: Model Efficiency

4. **Calculate computational cost $C(W')$:** Establish the resource consumption known as FLOPs in the pruned model. A lower operating FLOP count remains the key objective when conducting this optimization work on original models.
    - *Example:* A pruned BERT model may reduce FLOPs from 50 billion to 35 billion.
5. **Verify accuracy:** accuracy threshold for pruned models is defined by $A(W') \geq A(W) - \epsilon$.
    - *Example:* If the original accuracy is 90% and $\epsilon = 1\%$, the pruned model's accuracy must remain at or above 89%.

    By applying $\epsilon = 1\%$ to a 90% accurate model the maintained minimum acceptance threshold should still be at or above 89%.

Step 3: Inference Efficiency

6. **Measure inference time $T(W')$:** The duration needed to execute an input needs evaluation. The goal remains to decrease this processing time behind acceptable accuracy thresholds.
    - *Example:* Basic BERT algorithms process one input within a spam duration of 50 milliseconds. The inference time decreased to 30ms after implementing pruning.

Step 4: Sustainability

7. **Compute energy consumption $E(W')$:** NVIDIA Power Profiler tools enable users to measure system energy consumption. The system will reduce energy consumption while upholding the accuracy requirements.
    - *Example:* Pruning reduces energy use from its initial value of 100 Joules down to 70 Joules.

Step 5: Optimal Pruning Criteria

8. **Optimize pruning ratio p:** Sequential minor adjustments of the p parameter should occur when starting pruning ratios produce unacceptable performance drops.
    - *Example:* Lower the pruning ratio value to p =0.2 after $A(W')$ falls beneath $A(W) -$ and measure the results.

Step 6: Multi-Objective Optimization

9. **Simultaneous optimization:** The combination of multi-objective optimization and NSGA-II enables the minimization of $C(W')$, $T(W')$, and $E(W')$ while ensuring $A(W') \geq A(W) - \epsilon$.
    - *Example:* For a trade-off scenario:
        - **Option 1:** FLOPs = 35B, accuracy = 89.5%, inference time = 32ms, energy = 75J.

- **Option 2:** FLOPs = 33B, accuracy = 89%, inference time = 30ms, energy = 70J.

The deployment-focused selection process should return the configuration which maximizes performance while achieving optimal settings.

Final Output

10. **Return pruned model W':** The final pruned model W'should output together with accuracy measurement $A(W')$and power measurements $C(W')$, $T(W')$ and $E(W')$.

The organized methodology proves that this algorithm efficiently cuts down processing overhead while maintaining peak performance and sustainability standards.

Algorithm 1: Efficient Model Pruning Optimization Algorithm		
Input	**:**	• Original model weights W • Accuracy of the original model $A(W)$ • Acceptable accuracy loss threshold ϵ • Original computational cost $C(W)$ • Original inference time $T(W)$ • Original energy consumption $E(W)$ • Target pruning ratio p(optional, can be tuned)
Output	**:**	• Pruned model weights $'W'$ • Pruned model's accuracy $A(W')$ • Optimized computational cost $C(W')$ • Optimized inference time $T(W')$ • Optimized energy consumption $E(W')$
Step 1		**Model Pruning**
	I	**Initialize:** Start with the original weigh W.
	II	**Set target pruning ratio:** Choose a target pruning ratio p, where $0 \leq p \leq 1$
	III	**Prune weights:** • Compute the pruned weights W' by removing $p \cdot 100\%$ of the least important weights from W.

- $p = \frac{|\mathbf{W}-\mathbf{W}'|}{|\mathbf{W}|}$

Step 2 **Model Efficiency**

IV **Calculate computational cost:**

- Measure the computational cost C(W′) of the pruned model (e.g., using FLOPs).
- Objective: Minimize C(W′).

V **Verify accuracy:** Ensure that the pruned model's accuracy A(W′) satisfies:

- $A(\mathbf{W}') \geq A(\mathbf{W}) - \epsilon$, where ϵ is a predefined acceptable loss.

Step 3 **Inference Efficiency**

VI **Measure inference time:** Compute the inference time $T(W')$ of the pruned model.

- Objective: Minimize $T(W')$ subject to $A(\mathbf{W}') \geq A(\mathbf{W}) - \epsilon$.

Step 4 **Sustainability**

VII **Compute energy consumption:** Measure the energy consumption $E(W')$ of the pruned model.

- Objective: Minimize $E(W')$ subject to $A(\mathbf{W}') \geq A(\mathbf{W}) - \epsilon$

Step 5 **Optimal Pruning Criteria**

VIII **Optimize pruning ratio p:** If necessary, adjust p iteratively to balance performance and pruning efficiency.

- Objective: Maximize p such that $A(\mathbf{W}') \geq A(\mathbf{W}) - \epsilon$

Step 6 **Multi-Objective Optimization**

IX **Perform multi-objective optimization:**

- Simultaneously minimize $C(\mathbf{W}')$, $T(\mathbf{W}')$, and $E(\mathbf{W}')$ subject to:

$$A(\mathbf{W}') \geq A(\mathbf{W}) - \epsilon$$

X **Return pruned model:** Once an optimal trade-off is reached, output the pruned model W'.

The algorithm presents an optimization method for multi-objective pursuits that prunes models while sustaining accuracy standards while lowering computational expenses and decreasing inference periods combined with energy optimization goals. The optimization of the pruning ratio p occurs according to the particular requirements from the specific application domain. The deployment of large-scale transformer models depends significantly on efficient model pruning optimization algorithms which shrink complexity and computational requirements. The practical implementation of deep learning solutions

throughout different environments depends heavily on this algorithm as AI demand for efficient solutions continues to expand.

3.5 Ensemble of Winning Tickets Method

The Lottery Ticket Hypothesis (LTH) suggests that within large over-parameterized neural networks, specific subnetworks, known as "winning tickets," emerge naturally and can be trained independently to match the performance of the original model. Recent studies have explored various methods for leveraging multiple pruned subnetworks to enhance model performance and robustness, frequently interpreted using the LTH and ensemble techniques Zheng et al. [236].Another study by [24] introduced structured winning tickets at intial BERT training and Kobayashi et al. [237] discovered four winning tickets during the fine-tuning process. Gong et al. [238] applied the L1 distance method to regularize fine-tuning and explored the structure of the discovered subnetworks. Study [182] displays an implementation of this technique by applying lottery ticket hypothesis tactics for attention heads pruning during machine translation research. In this work, we leverage the LTH for iterative pruning of attention heads., identifying winning tickets by retaining only the most critical attention heads from a BERT model. This method refines the subnetwork structure iteratively and integrates these pruned subnetworks into a robust ensemble.

Figure 4 shows a generalized ensemble method framework where this study optimizes BERT model through application of winning ticket subnetwork pruning. Using a fully trained BERT network allows iterative pruning to extract pruned subnetworks keeping performance at equal levels. These smaller BERT models called winning tickets are combined into an ensemble Mohammend et al. [183], enhancing efficiency and performance over the original model. The ensemble pruning technique selects essential BERT attention heads for implementation through careful identification and retaining informative attention heads based on the lottery ticket hypothesis.

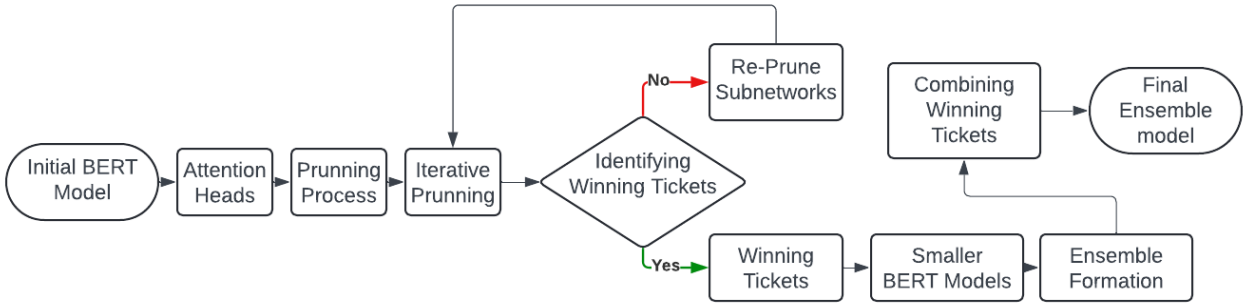


Figure 4: steps of optimizing BERT models through an ensemble of pruned subnetworks [42]

3.5.1. Dataset and Preprocessing

We utilized the *SST-2* and *CoLA* datasets from the GLUE benchmark for our experiments, dividing and shuffling the datasets using different random seeds to ensure that the model see the data in different order, with the BERT model checkpoint "bert-base-uncased" employed. The *SST-2* dataset serves as a binary sentiment dataset for detecting positive or negative sentiment at the sentence level in analysis. The *SST-2* dataset works without processing and functions widely throughout NLP tasks to maintain consistent well-structured sentences with valid labels. For the *SST-2* dataset, the training set was slightly reduced to 40,000 shuffled samplesthe validation dataset consisted of 800 instances, while the test set contained 1,500 instances reason being to allow fast experimentation, to minimise the requirements of computational resources, including memory, processing power, and time. For the *CoLA* dataset which is used for linguistic acceptability classification to tests a model's understanding of linguistic rules. The *CoLA* dataset is well-organized, with grammatically acceptable and unacceptable sentences already labeled and has been done by linguistic experts, to ensure quality and reliable data. For our experiments the training set contained 8,000 shuffled samples, the validation set included 1,000 samples, and the test set consisted of an equal number, 1,000 samples. This specific splitting and shuffling ensured that our model training and evaluation were based on diverse and representative data samples.

Input Representation

Both datasets require tokenized sentences as input. The input format for BERT is structured as: Input $= [\text{ CLS }] + \text{ Token}_1 + \text{ Token}_2 + \cdots + \text{ Token}_N + [\text{ SEP }]$

- **[CLS]**: A classification token, used for downstream classification tasks.
- **[SEP]**: This special token functions as an indicator which separates each sentence group.

For classification tasks that involve single sentences, like *SST-2* and *CoLA*, the input structure is defined as: Input $= [\text{CLS}] + \text{ Sentence } + [\text{SEP}]$

3.5.2. Initialization of the Model and Tokenizer

Our sequence classification operations with BERT models began by initializing them together with their tokenizer from pretrained checkpoints. During this step our model retrieves knowledge from BERT's extensive pretrained architecture while applying transfer learning principles. For sequence classification tasks BertTokenizer (BertTokenizer) from the Hugging Face Transformers library associated with bert-base-uncased combined with the BERT model delivered suitable tools for our experimental needs by enabling efficient and accurate sequence classification tasks. The following section explains these processing stages while breaking down their mathematical aspects.

1. Loading the Pre-trained Model

The process begins by loading **BERT Base model,** consisting of:

12 Transformer block of layers: model's internal structure extracts information at successively higher levels of abstraction.

- **768 hidden units per layer:** Each embedding unit in the token representation contains 768 dimensions.
- **12 self-attention heads per layer:** Each head captures different aspects of word relationships.

- **Total Parameters:** Approximately 110 million.

This model was initially trained on an extensive text corpus using two primary tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), as applied by Google. In the pre-training stage the model establishes word contextual understanding through the analysis of surrounding text.

2. Initializing Parameters

Once the model is loaded, next is initiation which involves the processing of model parameter values. The parameters include:

- **Embedding Weights** (W_e): It does the mapping of the tokens to dense vector representations.
- **Attention Layer Weights** (W_q, W_k, W_v): Simplify the mathematical procedures for self-attention by establishing definitions for query and key parameters with their associated value function.
- **Feed-Forward Layer Weights** (W_{ff}): The attention mechanism produces its results through the weights W_e, W_q, W_k, W_v which form part of the attention layer.

Mathematically, this initialization is represented as:

$$\{W_e, W_q, W_k, W_v\} \leftarrow \text{ Pre-trained weights} \tag{92}$$

Quantities of well-learned weights obtained during extensive pre-training are used within BERT initialization to maintain their application.

3. Attention Mechanism Initialization

By implementing its self-attention mechanism BERT can fully recognize contextual relationships throughout sequential input text. Key mathematical steps include:

Input Representation: A BERT model maps every token across a sequence with n tokens into vector form x_i:

$$X = [x_1, x_2, \dots, x_n] \tag{93}$$

Linear Transformations: Learning weights transform input tokens into three types of components which are **queries** ($\boldsymbol{Q}$), **keys** ($\boldsymbol{K}$), and **values** ($\boldsymbol{V}$) using learned weights:

$$Q = XW_q, K = XW_k, V = XW_v \tag{94}$$

Scaled Dot-Product Attention: The attention calculations produce scores through the dot product operation between $\boldsymbol{Q}$ and $\boldsymbol{K}$ while scaled by the square root of the key dimension d_k:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{95}$$

Concatenation of Heads: The completed attention output emerges from concatenating all attention head outputs and following them with a linear transform.

Example: An Initialization with a Simple Sentence

When using the input text "I love NLP," Therefore the initialization would look like this:

Tokenization: The input data is processed into numerical IDs by the tokenizer, e.g., $[101,1045,2293,17953,102]$ where $[101]$ is $[CLS]$, and $[102]$ is $[SEP]$.

Embedding Layer: Embeddings demonstrate W_e to convert tokens into vectors which create a dense sequential output

Weight Initialization: Pre-trained weights (W_q, W_k, W_v) are used to initialize the attention layers**.**

For example, queries (Q), keys (K), and values (V) for "I" are computed as:

$$Q = XW_q, K = XW_k, V = XW_v \quad (96)$$

Attention Calculation: The system determines using attention scores what degree "I" needs to focus on "love" combined with "NLP."

Output Processing: The output from attention is processed through the feed-forward layers to produce contextualized representations.

During BERT model initialization it involves setting up pre-trained weights for embeddings, attention layers, and feed-forward layers. Model initialization enables pre-trained knowledge to form a sturdy foundation which enabling effective fine-tuning for specific downstream tasks.. The mathematical basis of the attention model within BERT enables sequence relationship detection to achieve outstanding measurable performance throughout natural language processing tasks.

3.5.3. Ensemble of Winning Tickets

The winning ticket ensemble required initial setup as an empty set. The model required a mask matrix containing all values set to 1 to facilitate attention head pruning. The essential components of BERT architecture are identified and retained by the mask matrix. The BERT architecture served as a basis to initialize a random ticket / a sub-network while using the specified checkpoint. Adam performed as optimizer together with cross-entropy as loss function when training the sub-network. Tailored configurations enabled both efficient training and optimization of the sub-network to obtain results that matched the model objectives.

- The number of sub-networks (n) for the ensemble is defined.
- An empty set $(T = \{\})$ is initialized to store the **winning tickets**.

Ensemble Initialization: Ensemble initialization functions as a foundational step for creating neural network ensembles under LTH implementation. LTH suggests that randomly initialized neural networks with numerous weights harbor compact, trainable structures

known as "winning tickets" which match the performance of the entire system. Ensemble initialization requires determining sub-network counts along with storing them for future ensemble prediction. The procedure is outlined in detail as below.

Setting Sub-network number $(\boldsymbol{n})$: The ensemble's configuration begins with selecting the number of winning tickets which will become its sub-networks. The number (n) of sub-networks depends on both ensemble diversity needs and computational resource capacity. Let the value of $n = 10$ to automatically generate an ensemble composed of ten sub-networks. Mathematically denoted as: $n \in \mathbb{N}^+$

In which $\mathbb{N}^+$ denotes set of positive integers.

Initializing the Set for Winning Tickets $(\boldsymbol{T})$: The initialization step begins with an empty set T which functions to store winning tickets. Each successful winning ticket signifies a pruned sub-network holding particular parameters (θ_i).

This initialization is denoted as: $T = \{\}$

At this initialization step T contains no elements but will later receive parameters from identified winning tickets which follow in subsequent steps.

Example of Initialization Process

Define the Number of Sub-networks: Let's say we choose $n = 5$. This means that we develop an ensemble model consisting of five sub-networks.

Initialize the Set: The winning ticket storage set $T = \{\}$ will start empty.

Identify and Store Winning Tickets: Through iterative training and pruning methods winning tickets get identified. For example:

Winning Ticket 1: θ_1

Winning Ticket 1: θ_2

Winning Ticket 1: θ_3

Winning Ticket 1: θ_4

Winning Ticket 1: θ_5

After identifying these tickets, the set T is updated as: $T = \{\theta_{1,}, \theta_2, \theta_3, \theta_4, \theta_5\}$

Mathematical Representation of the Ensemble: The ensemble becomes ready for making predictions when winning ticket predictions are stored. For an input x, the ensemble output $E(x)$ is the average of predictions from all n sub-networks:

$$E(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x; \theta_i) \quad (97)$$

Where:

- $E(x)$: The ensemble's output for input x.
- $f_i(x; \theta_i)$: The i-th sub-network's prediction using its parameters θ_i.

Ensemble initialization begins the winning ticket ensemble process through its setup of sub-network count (n) and an empty ticket storage set (T) while preparing for subsequent pruning and training steps. The initialization stage sets the foundation for the ensemble to harness Lottery Ticket Hypothesis benefits which generate a strong compact model through the aggregation of individual winning tickets. Mathematical structures defined here explain ensemble procedures that establish foundation blocks for efficient and accurate ensemble predictions.

Mask Matrix Initialization: Each sub-network iteration i begins with a M_i binary mask matrix serving to control which attention heads are retained. At the beginning of the process all elements in the mask receive a value of 1 to show that pruning has not commenced.

Mathematical Representation: $M_i = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$

Here:

- h: Number of attention heads.
- m: Number of layers in the BERT model

Sub-Network Initialization: A sub-network T_i obtains its initial structure by matching with the BERT structure. Initialization follows a randomized approach while respecting BERT model structure to maintain basic foundational capacity. Mathematical Representation: $T_i = f(x; \theta_i)$

Where:

- θ_i: Parameters of the sub-network initialized randomly.
- f: Function representing the BERT model.

Training the Sub-Network: With complete training dataset D_t and a specified number of epochs or until convergence the sub-network T_i is trained on that particular dataset. Through training we adjust θ_i to minimize a specific predefined loss function. Mathematical Representation:

$$\mathcal{L}(T_i, D_t) = \frac{1}{N} \sum_{j=1}^{N} \text{Loss}\left(T_i\left(x_j\right), y_j\right) \quad (99)$$

Where:

- N: Number of training samples.
- x_j: Input features for the j-th sample.
- y_j: Corresponding label for x_j.
- $Loss$: The loss function (e.g., cross-entropy for classification).

Evaluation: Sub-network undergoes evaluation on dataset D_v to verify its performance by meeting pre-established criteria such as accuracy threshold. Mathematical Representation: $\text{Performance}(T_i, D_v) = \text{Accuracy}(T_i, D_v)$

Where:

$$\text{Accuracy } = \frac{1}{M}\sum_{k=1}^{M} \mathbb{I}(T_i(x_k) = y_k) \quad (100)$$

Here:

- M: Number of validation samples.
- $\mathbb{I}$: A function that returns 1 for a correct prediction and 0 for an incorrect one.

Pruning: Performance criteria define when pruning begins because it deletes less vital attention heads from the sub-network. The mask matrix M_i transforms based on this pruning operation. The pruning process updates the mask matrix M_i through its new values. Mathematical Representation:

$$M_i' = M_i \odot \text{ Pruning_Mask} \quad (101)$$

Where:

Pruning_Mask : contains binary values that show which attention heads should remain intact (1) or be removed (0). The output network containing the preserved attention heads takes this form:

$$T_i' = f(x; M_i' \odot \theta_i) \quad (102)$$

Adding to Ensemble: The ensemble receives the successful sub-network T_i' after its pruning process ends. The final ensemble set T thus includes all successful sub-networks. Mathematical Representation:

$$T = T \cup \{T_i'\} \quad (103)$$

The ensemble set T gathers all sub-networks obtained from pruning operations.

Two key operations occur when identifying winning tickets: the initial sub-network receives full capacity before training and evaluation while pruning unimportant attention heads followed by inclusion of the pruned sub-networks into a collective ensemble. Through this process smaller efficient sub-networks can be created until they belong to an ensemble which combines their collective strengths for optimal performance.

3.5.4. Aggregating Predictions in Ensemble Learning

The critical aspect of ensemble learning particularly when using the lottery ticket hypothesis entails predictions aggregation. The process merges prediction results from diverse "winning tickets" (truncated subnetworks) in order to create the final output. Combination of predictions from multiple winning tickets improves both the test performance by increasing accuracy and enhances robustness on a test dataset. The following structured breakdown outlines this method alongside mathematical examples and related formulations.

Prediction by Each Ticket: The test dataset D_t contains each sample x and these undergo evaluation through all ensemble tickets T_i in sequence. The prediction output P_i from each ticket can appear as class probability distribution or logits values or direct class predictions based on the targeted task (classification or regression).

Mathematical Representation: $P_i = \theta_i(x)$

Each winning ticket's prediction function for input x is described by the notation $\theta_i(x)$.

Example: Consider three winning tickets (T_1, T_2, T_3) and a test sample x. Predictions might look like this:

An example describes the situation with three winning tickets (T_1, T_2, T_3) together with a test data point x. Predictions might look like this:

- $P_1 = \theta_1(x) = [0.7,0.2,0.1]$ (class probabilities for $\mathrm{A}, \mathrm{B}, \mathrm{C}$)

- $P_2 = \theta_2(x) = [0.6,0.3,0.1]$
- $P_3 = \theta_3(x) = [0.8,0.1,0.1]$

Aggregating Predictions: The model generates the final prediction $P(x)$ through an aggregation of all winning ticket predictions. Mathematical Representation using the averaging method:

$$P(x) = \frac{1}{n}\sum_{i=1}^{n} \theta_i\ (x) \quad (104)$$

Final prediction involves dividing the sum of winning ticket predictions by the n value which represents the total number of winning tickets.

Example: Using the predictions from $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$:

$$P(x) = \frac{1}{3}(P_1 + P_2 + P_3) \quad (105)$$

Calculating for each class:

Class A: $\frac{0.7+0.6+0.8}{3} = 0.7$

Class B: $\frac{0.2+0.3+0.1}{3} = 0.2$

Class C: $\frac{0.1+0.1+0.1}{3} = 0.1$

Thus, the aggregated prediction is: $P(x) = [0.7,0.2,0.1]$

Decision Making: The most probable category from $P(x)$ determines the final class assignment of x. Mathematical Representation: $Final\ Class = argmax(P(x))$

Example: $From\ P(x) = [0.7,0.2,0.1]$: $Final\ Class = argmax([0.7,0.2,0.1]) = A$

Combining prediction results from successful tickets brings together distinct capabilities of its separate components to boost overall system performance. The ensemble produces

reliable and accurate test results through prediction averaging methods demonstrated in the previous example. Using this step, we can fully leverage ensemble learning capabilities in relation to the lottery ticket hypothesis.

Iterative Optimization: Iterative optimisation occurs when sub-network generation combined with pruning and evaluation happens consecutively until the ensemble reaches its required iterations to achieve refinement. The EWT approach receives final assessment through aggregated performance metrics.

3.5.5. Training Configuration

Proper selection of training parameters allowed for optimal learning process execution. Training comprised five iterations using parameters of a learning rate at 2e-5 combined with a batch size of 16. The established convergence threshold value stood at 0.001 to guarantee stable training operations. The training arguments consisted of saving the pruned model and best model loading method at training completion and three training epoch limit definition and defining the evaluation and saving strategies. We restricted the evaluation process to use F1 score and accuracy alongside a maximum limit of 10 saved models. The model was configured to keep training configurations efficient while building robustness so it could avoid overfitting while reaching maximum possible performance.

3.5.6. Selection of Informative Attention Heads

A saliency score functions as a performance-based metric to evaluate the significance of model components including neurons and features as well as attention heads. Saliency scores derive their values from sensitivity computations which operate against model inputs and parameters [239]. The saliency scores in BERT attention-based models identified the relevancy between model output and input parts through evaluation of attention heads [240].

Let H_i denote the ith attention head. The saliency score for attention head H_i denoted as S_i were computed based on its impact on the model's loss or output. We measured the change in the model's loss L when H_i is perturbed or masked:

$$S_i = L(f(x, mask(H_i))) - L(f(x)) \quad (106)$$

Here, mask H_i represents a modification of the model where the ith attention head is masked or zeroed out.

3.5.6.1. Importance Score Calculation

The calculation of importance scores represents the initial stage of attention head pruning because it measures how much each head assists model performance. Common methods include:

Sensitivity-Based Importance: When performing sensitivity analysis the attention head h importance score I_h depends on the expected sensitivity of the model to the head-specific mask variable ξ_h. Mathematically:

$$I_h = \mathbb{E}_{x \sim X}\left[\frac{\partial L(x)}{\partial \xi_h}\right] \quad (107)$$

Where:

- $L(x)$: Loss function for input x,
- X: Data distribution,
- ξ_h: Binary mask (1 for active, 0 for pruned).

A score calculated from this assessment reveals the degree of model performance change when a specific head is disabled. The model performance relies more heavily on a given head when the score reaches higher values.

Layer-wise Relevance Propagation (LRP): serves as another method to determine attention head importance through calculations. The assessment method determines the individual head contributions to the output results through analysis of the gradients flowing back through the network. Among different methods there exists a concept called importance score that delivers key information defined as:

$$I_h = \sum_{i=1}^{n} \text{Relevance}(h_i) \quad (108)$$

Where:

- h_i: Output of the i-th attention head,
- Relevance: Contribution of the head to the model output.

This approach computes importance based on gradient flow through the network.

3.5.6.2. Pruning Strategies

The process of attention head pruning starts through importance scoring followed by implementations of iterative pruning or Lottery Ticket Hypothesis methods.

Iterative Pruning: This method enables the model to adapt through successive pruning of attention heads with the lowest importance scores in multiple stages. Steps include:

- **Calculate Importance Scores**: The procedure computes importance values through sensitivity analysis or LRP methods.
- **Sort and Select Heads**: A sorting process must be applied to attention heads followed by the selection of the most vulnerable elements according to assigned scores.
- **Prune Heads**: The model performs removal of the selected attention heads.
- **Fine-Tune**: The model receives additional training to regain performance after pruning.

Mathematically, iterative pruning represented as: $\hat{T} = T - \{h \in T : I_h < \theta\}$

Where:

- T: Set of all attention heads,
- $\hat{T}$: Remaining heads post-pruning,
- θ: Importance score threshold

3.5.7 Practical Calculations and Algorithms for Identifying Winning Tickets

In this sub-section, we specify the steps in detail and the algorithm that was used to carry out the Ensemble of Winning Tickets, which particularly helps in reducing the number of attention heads of BERT more efficiently while maintaining the model's performance. The primary objective is to train the subnetworks of the BERT model and prune them, searching for possible subnets where their performance can be equal or higher than the performance of the original model. Here we describe methods which make use of independent pruned models to enhance the final model performance by using model ensemble technique. The following algorithm 1 provides developers with a systematic method to build winning tickets using training and validation data for achieving top performance results. The methodology concludes in aggregating multiple simplified subnetwork predictions using a basic averaging method to develop robust dependable end results. BERT alongside other large-scale models benefits from better runtime and improved effectiveness when this normalization approach is implemented.

Algorithm 1: Ensemble of Winning Tickets Approach for BERT

Input :

- A pre-trained BERT model: M
- Number of ensembles to be created: n
- Validation dataset: D_v
- Test dataset: D_t

Output :

- An ensemble of winning tickets: T

Steps

1 Load the pre-trained BERT model: $\boldsymbol{M}$

2 Initialize an empty ensemble set to store the winning tickets: $\boldsymbol{T} = \{\}$

3 Set the number of ensembles to be created: $\boldsymbol{n}$

4 For each ensemble iteration $\boldsymbol{i}$ from 1 to $\boldsymbol{n}$:

1. Initialize a mask matrix for the attention heads with all values set to 1: $M_i = 1_m \times d$

2. Randomly initialize a sub-network (ticket) based on the BERT architecture: $T_i = random_init(M_i)$
3. Train the sub-network for a fixed number of iterations or until convergence on a training dataset: $T_i = train(T_i, D_t)$
4. Evaluate the sub-network's performance on the validation dataset: $T_i = evaluate(T_i, D_t)$
5. If the sub-network meets the desired performance criteria:
    - Apply the mask matrix to the sub-network, effectively pruning the selected attention heads: $T_i = prune(T_i, M_i)$
    - Add the pruned sub-network (winning ticket) to the ensemble set: $T = T \cup T_i$

5 For each sample x in the test dataset:

- Feed the sample through each winning ticket in the ensemble set:
$$P_i = T_i(x)$$
- Aggregate the predictions from each winning ticket to obtain the final prediction:
$$P(x) = f(\theta_1(x), \theta_2(x), \dots, \theta_n(x)) = \frac{1}{n}\sum_{i=1}^{n} \theta_i(x)$$

6 Evaluate the performance of the ensemble on the test dataset using appropriate metrics: $M = evaluate(T, D_t)$

7 Perform steps 4-6 iteratively until the desired number of iterations or ensembles is reached.

8 Aggregate the performance metrics across all ensembles to assess the overall performance of the ensemble method: $M = aggregate_metrics(T)$

Notation

- Ensemble of Winning Tickets E: Combining multiple pruned models M_i where $i = 1,2, \dots, n$.

$$E(x) = f(T_1(x), T_2(x), \dots, T_n(x)) \quad (109)$$

- Aggregation Function f: Aggregates the predictions of the pruned models using simple averaging.

$$P(x) = f(\theta_1(x), \theta_2(x), \dots, \theta_n(x)) = \frac{1}{n}\sum_{i=1}^{n} \theta_i(x) \quad (110)$$

Process of forming an ensemble.

- Form the Ensemble Set T:

$$T = i = \bigcup_{i=1}^{n} (prune(\theta_i, 1) \mid evaluate(\theta_i, D_v))$$

- Where:

$$\theta_i = train(init(1), D_t) \quad (111)$$

- Make Predictions with Aggregation:
  - Obtain individual predictions from each pruned sub-network:

$$\theta_i(x) for\ i = 1,2, \dots, n \quad (112)$$

  - Aggregate these predictions using the aggregation function f:

$$P(x) = f(\theta_1(x), \theta_2(x), \dots, \theta_n(x)) = \frac{1}{n}\sum_{i=1}^{n} \theta_i(x) \quad (113)$$

The outlined algorithm demonstrates how to create an ensemble from pruned attention heads of BERT models while performing aggregated predictions. The adopted procedures boost both result reliability and reproducibility. Our research uses *SST-2* and *CoLA* datasets from GLUE benchmark as solid bases together with BERT model which represents a typical sequence classification method and further strengthens its performance by utilizing pre-trained checkpoints.

A winning tickets ensemble strategy focuses on important components of BERT architecture to enhance model operational efficiency. The designed training procedures achieve maximum learning performance while preventing overfitting behavior to ensure solid findings. The test of importance scores enables effective transformer model simplification because it follows the Lottery Ticket Hypothesis framework. Models reach optimal performance-efficiency ratios when they choose appropriate heads to retrain. The mathematical descriptions guide the process of computing importance scores together with the pruning methods and performance evaluation.

3.5.8. Performance Metrics

Accuracy Calculation of Sentiment Analysis on *SST-2* dataset: The model received training on *SST-2* data before performing validation procedures on its test results. An example shows how the model performs its prediction process: The model correctly identified 400

positive sentiments within the dataset (this scenario represents True Positives). The predictions identified 350 negative sentiments in the dataset with accuracy (represents True Negative). The model produced 50 False Positives by incorrectly identifying negative sentiments as positive. The model incorrectly detected 30 negative sentiments among the test data set (False Negatives). The following formula established the calculation method for accuracy:

$$\text{Accuracy } = \frac{TP+TN}{TP+TN+FP+FN} \tag{114}$$

Calculation:

$$\text{Accuracy } = \frac{400+350}{400+350+50+30} = \frac{750}{830} \approx 0.9036 \text{ or } 90.4\% \tag{115}$$

Accuracy calculation on Linguistic Acceptability on *CoLA* dataset: Predictions made by the model: Inaccuracies occurred with 600 True Positive cases (proper identification of acceptable sentence decisions) during the prediction process on the *CoLA* dataset. The 300 predictions that indicated unacceptable sentences were classified correctly as True Negatives. The model incorrectly identified twenty instances (False Positive) as acceptable while the actual sentences were unacceptable. The model classified 80 sentences wrongly by projecting them to have acceptable sentences although they were actually unacceptable (False Negative).

Accuracy Formula:

$$\text{Accuracy } = \frac{TP+TN}{TP+TN+FP+FN} \tag{116}$$

Calculation:

$$\text{Accuracy } = \frac{600+300}{600+300+20+80} = \frac{900}{1000} = 0.9 \text{ or } 90\% \tag{117}$$

F1 score Calculation of Sentiment Analysis on *SST-2* dataset: There were several predictions established by the created model. The model achieved 40 True Positive results when correctly predicting positive sentiments in the dataset. The correct identification of negative sentiments produced 30 True Negatives. The prediction of positive sentiments included 10 instances where the system made mistakes (False Positive). The evaluation shows that 20 cases were classified incorrectly as negative sentiments when they should have been identified as positive sentiments (False Negative).

Step 1: Calculate Precision and Recall

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{40}{40+10} = 0.80 \quad (118)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{40}{40+20} \approx 0.67 \quad (119)$$

Step 2: Calculate F1 Score

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision + Recall}} = 2 \times \frac{0.8 \times 0.6667}{0.8+0.6667} \approx 0.74 \quad (120)$$

The F1 score represents a trade-off between precision and recall, reflecting the model's overall performance at ~73.68%.

F1 score calculation of Linguistic Acceptability on *CoLA*: Model predictions indicate that there are 50 true positives together with 40 true negatives yet the system detects 5 false positives and generates 10 false negatives.

Step 1: Calculate Precision and Recall

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{50}{50+5} \approx 0.91 \quad (121)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{50}{50+10} \approx 0.83 \quad (122)$$

Step 2: Calculate F1 Score

$$\text{F1 Score } = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{0.9091 \times 0.8333}{0.9091 + 0.8333} \approx 0.87 \quad (123)$$

The F1 score demonstrated a well-maintained equilibrium between precision and recall, with a performance of ~86.96%.

Inference Speed Calculation of Sentiment Analysis on *SST-2* data: We used fine-tuning on the *SST-2* data before measuring the model's validation set inference duration. Given the following: The model required ten seconds to analyze one thousand sentences during inference. Inference Speed Calculation:

$$\text{Inference Speed } = \frac{\text{Total Sentences}}{\text{Total Inference Time}} = \frac{1000 \text{ sentences}}{10 \text{ seconds}} = 100 \text{ sentences / second} \quad (124)$$

This means that our model processes 100 sentences per second during inference.

Inference Speed Calculation of Linguistic Acceptability on *CoLA* dataset: We measured an evaluation timeframe of 8 seconds while using the *CoLA* dataset against this model. The model needed 8 seconds to run 500 sentences in total. Inference Speed Calculation:

$$\text{Inference Speed } = \frac{\text{Total Sentences}}{\text{Total Inference Time}} = \frac{500 \text{ sentences}}{8 \text{ seconds}} = 62.50 \text{ sentences/second} \quad (125)$$

During inference this model handles 62.5 sentences each second. Developers use these computations to determine the operational performance of NLP models particularly within environments that need speedy inference and minimal resources.

Memory Footprint Calculation of sentiment analysis for *SST-2* dataset: The calculation of memory footprint depends upon model parameters together with input data and all intermediate activation values from inference processing. The total storage size needed for BERT-base model parameters amounts to about 440 megabytes because each of its 110 million parameters occupies 4 bytes (32 bits) for storage.

$$\text{Memory for Parameters } = 110{,}000{,}000 \times 4 = 440{,}000{,}000 \text{ bytes } \approx 440\text{MB} \quad (126)$$

The model accepts 128 tokenized sentences through BERT's maximum length while each token contains a 768-dimensional vector the input memory is:

$$\text{Memory for Input } = 128 \times 768 \times 4 = 393{,}216 \text{ bytes } \approx 0.39\text{MB} \quad (127)$$

During inference the memory requirement matches the total memory need for maintaining network parameters. Memory for Activations $\approx 440\text{MB}$

Total Memory Footprint:

$$\text{Total Memory } = 440\text{MB} + 0.39\text{MB} + 440\text{MB} \approx 880.39\text{MB} \quad (128)$$

Memory Footprint Calculation of Linguistic Acceptability for *CoLA* dataset: The same BERT-base model with 110 million parameters: Memory for Parameters $= 440\text{MB}$**.** The 64 tokens per input sentence (*CoLA's* max limit) requires processor memory of 196,608 bytes which equals approximately 0.19MB. Memory for Input $= 64 \times 768 \times 4 = 196{,}608$ bytes $\approx 0.19\text{MB}$

Intermediate Activations: Memory usage during inference: Memory for Activations $\approx 440\text{MB}$

Total Memory Footprint: Total Memory $= 440\text{MB} + 0.19\text{MB} + 440\text{MB} \approx 880.19\text{MB}$

Energy Consumption Calculation of sentiment analysis for *SST-2* dataset: The calculation of energy usage for a designated task requires the following formula: Energy $=$ Power $\times$ Time

Where the hardware utilizes Power to determine its average energy consumption measured in watts. The parameters for inference are: The GPU utilizes an average power of 200 watts for its operations. Inference Time is 10 seconds for processing 1,000 sentences.

Step 1: Convert inference time from seconds to hours:

$$\text{Time} = \frac{10 \text{ seconds}}{3600 \text{ seconds / hour}} \approx 0.00278 \text{ hours} \quad (129)$$

Step 2: Calculate energy consumption:

$$\text{Energy} = 200 \text{ watts} \times 0.00278 \text{ hours} \approx 0.56 \text{ watt-hours (Wh)} \quad (130)$$

Energy Consumption Calculation of Linguistic Acceptability for *CoLA* dataset

The parameters for inference are:

- **Power Consumption**: 200 watts (same GPU).
- **Inference Time**: 8 seconds for processing 500 sentences.

Step 1: Convert inference time from seconds to hours:

$$\text{Time} = \frac{8 \text{ seconds}}{3600 \text{ seconds / hour}} \approx 0.00222 \text{ hours} \quad (131)$$

Step 2: Calculate energy consumption:

$$\text{Energy} = 200 \text{ watts} \times 0.00222 \text{ hours} \approx 0.444 \text{ watt-hours (Wh)} \quad (132)$$

3.5.9. Tools and frameworks used for measuring performance.

Through our implementation we used PyTorch deep learning frameworks to measure accuracy, F1 score, precision and recall through built-in tracking functions during training and evaluation. The Scikit-learn library presented various classification metrics through its functions that allowed users to compute accuracy, F1 score, precision and recall assessment. We accessed Weights & Biases to obtain one integrated solution that enabled experiment tracking alongside visualization tools for metrics during ML project collaboration. Machine learning evaluation requires accuracy and F1 score performance together with speed of inference execution and memory usage and power consumption metrics. Different metrics and tools need to be selected according to the particular application requirements together with their corresponding trade-offs. When resources are limited the requirement to minimize

memory consumption along with energy use surpasses the need for maximum accuracy achievement.

Chapter 4: Results and Discussion

The first part of this section delivers an extensive review of results and discussion through a thorough breakdown of the conceptual framework components together with their explanations. We display the experimental findings for the multi-objective optimization approach while conducting a performance review against state-of-the-art methods. The method underwent testing for pruning model comparison using *CoLA* and *SST-2* datasets benchmarked against existing pruning models. This *evaluation* examines all experiments and focuses on key metrics across all experiments, including accuracy, computational cost, inference time, and energy consumption. This study evaluated the interrelation between accuracy levels and resource efficiency at different pruning ratios where we assessed model performance benefits against optimizing resource efficiency. The experiments benefit from full reproducibility because researchers can access complete codebase details through the GitHub repository located at https://github.com/nksmarts/Efficient-Model-Pruning.git. This transparency facilitates validation and further exploration by the research community.

Finally, the performance along with efficiency of BERT model receives analysis relative to our ensemble model across both *CoLA* and *SST-2* datasets. This evaluation conducts a performance assessment between pruned ensemble models and baseline models to determine efficiency versus performance levels. The ensemble method implementation codes are accessible at GitHub through https://github.com/nksmarts/Ensemble-of-winning-tickets/blob/main/New_pruning_BERT_attention_heads_F1_score.ipynb.

4.1. Conceptual Framework Components

4.1.1 Sparsification Schemes

In this subsection, we discuss various sparsification schemes as illustrated in figure as a critical component of the pruning process in neural networks, focusing on both model sparsity and ephemeral sparsity. Here our findings highlight that, these sparsity techniques play a crucial role in improving the effectiveness of overparameterized models by selectively

removing weights, neurons, filters, channels, and heads. Another aspect is the effectiveness offered by ephemeral sparsity that adapts during the computation of specific examples, for instance, dropout, ReLU, and conditional computations. This flexibility plays an important role in real-time optimization, drawing neural networks into close alignment with practical applications.

Findings from the literature also correlates with [241] view on dropout as the form of ephemeral sparsity, but it generalizes on this concept by including other methods of dynamic sparsification such as conditional computations and gradients sparsity. The value of this work lies in the clear demonstration of how both model and ephemeral sparsity can improve neural networks for practical use. Pruning algorithms work quite well in practice by reducing the size of networks and the number of floating points needed at test time while improving the generalization of models, but a few points arise for future research for better pruning of networks: (i) How to tune the pruning schedules for specific tasks and network architecture?

4.1.2. Sparsification Schedules

Understanding the sparsification schedule represents a crucial element within the framework because it requires defining the appropriate time and method for sparsification implementation. Three fundamental training schedule classes exist for pruning operations: Train then Sparsify and Sparsify During Training and train-then-sparsify with regrowth [242]. The Train then Sparsify schedule method involves neural network convergence followed by sparsification application to the final model. The method allows optimal model performance to be achieved before sparsity is introduced to the network. The Sparsify During Training schedule includes using sparsification techniques through each training epoch in addition to iterative sparsification which performs gradual pruning across successive training epochs. The train-then-sparsify procedure includes the regrowth method where the network partially reconstructs its weight values without changing the desired sparsity ratio after pruning.

The different scheduling techniques can be positioned within generic scheduling procedures that allow exceptions of certain stages. The network structure receives its initial

configuration by either disk-loading a previous model or employing a framework or generating random weights or by implementing sparse construction through single-shot network pruning (SNIP). The network weights receive initial configuration which can happen randomly or with pre-trained weights together with sparse network-specific techniques such as synaptic flow. The network achieves training through a standard dense schedule or by employing sparsity-inducing schedules alongside regularization until necessary convergence occurs. The network components go through pruning and potential before the optional retraining step to improve accuracy rates. A repetitive application of the training/sparsification schedule with multiple iterations enables an improvement of final result quality. Weight values need resetting during the training period according to specific requirements. The training process contains various steps which lead to developing successful sparse networks. Research progress in artificial intelligence and neural network pruning depends on selecting optimal scheduling parameters with optimal sparsification methods to create efficient accurate models. A real-world deployment setting becomes available to utilize the model after successful fine-tuning completes.

4.1.3. Deployment

In this subsection we highlight key considerations for deploying pruned models from the conceptual framework in real-world applications, including measures such as ease of deployment, power consumption, and other costs. We underscore that pruning indeed not only decreases the size of the model and the computational cost but also makes the model easier to deploy on different platforms, including cloud servers, and mobile platforms. Additional support for the aspect of pruning comes from [231], where energy consumption by pruning methods was reduced during both operation and training, thus making the models more sustainable in energy-constrained environments. This is in line with the now trending environmentally sensitive AI systems development [243].

While the study's strength lies in its holistic view of deployment challenges, one limitation is the potential trade-off between real-time processing and maintaining high

accuracy. Surprisingly, it turned out that models pruned for cost and resource consumption can achieve real-time inference requirements, an area where more aggressive pruning could have been expected to hinder performance. The study sought to explain the possible key considerations that can be useful in determining how to employ pruned models or models with fewer parameters in real-world applications including resource use, conservancy, and practicality.

4.1.4. The Potential Impact of Model Optimization

This study examines the transformative effect of model optimization specifically on the diverse fields of application development, software practices, hardware compatibility, and academic research. Our key findings reveal that pruned models offer significant advantages in creating faster and more efficient AI applications that can be deployed at the mobile and edge-end computing paradigm discussed by [244]. Additionally, model optimization accelerates the software development process by reducing training times and iterations, resulting in a more streamlined development lifecycle. Furthermore, the overall software development process is also made faster by model optimization, reducing training times and iterations.

The evidence supports the premise that optimization is the key enabler of the progress in the general efficiency of AI in various sectors. From this perspective, the present research shares common views with [232] by underlining that further improvements on the model are critical for the development of new techniques in the AI field. However, we extend this by demonstrating the implications of this on hardware compatibility, specifically GPUs and TPUs, which significantly enhance performance and resource efficiency.

One of the study's strengths is that it looks at the way pruned models underpin innovation in architecture and apply it to AI work in areas including natural language processing and computer vision. However, there is still a limitation, achieving the optimal balance between model compactness and performance can be complex particularly when it comes to its deployment on various hardware. Unexpectedly, our work showed that model optimization is not only beneficial for resource savings but also for making AI technology more inclusive by making AI accessible on resource-constrained devices, creating new possibilities

for development. Investigation attempted to establish the various benefits concerning model optimization and its contribution towards the development of AI systems. The significance of this work is in showing how optimization makes a positive impact to the advancement of AI and makes it accessible to the entire populace across the globe as well as making a positive impact on nature and the way we live.

4.1.5. Model Pruning Verses Model Efficiency

Figure 5, which examines the relationship between pruning percentage verses model size reduction, presents an important pattern consistent with primary conclusions of prior analysis. As pruning percentages increase, model size reduction becomes more pronounced, particularly in complex architectures such as transformers. For example, the reduction of model size for 80% in Pruned NMT, resulted in the largest model size reduction of 510MB. This observation supports previous work by [242], where significant size reduction is illustrated with increases in pruning percentages, thus confirming the efficiency of pruning approaches. These results suggest that pruning is a viable strategy, which can be used to enhance the efficiency of deep learning models, and effectively implement them in in resource-constrained environments.

In support of this, [233] conducted a similar analysis showing that pruning methods resulted in models that are as much as 80% smaller than their original versions and without requiring additional complex adjustments. This shows the ability of pruning methods in shrinking sizes of models while retaining functionality hence the need for them in applications that require the use of small models with efficient resource. This pattern of size reduction through different pruning methods shows the practical benefits of pruning in optimizing computational resources, particularly in intricate models.

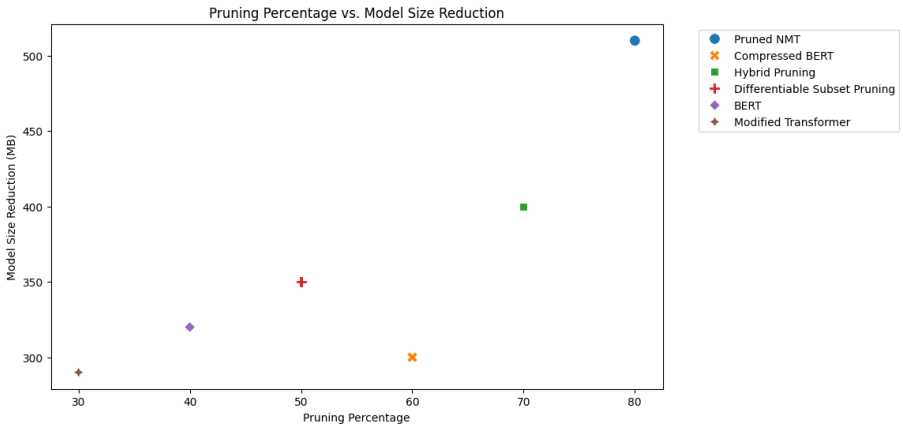


Figure 5: Pruning Percentage Verses Model Size Reduction

Figure 6, focuses on the relationship between pruning percentage and performance metrics including accuracy as well as BLEU score. The findings suggest a nuanced relationship, moderate pruning percentages levels ranging from 50-70% are seen to perform fairly well according to the performance indicator where two of the models are seen to have performed well, that includes the Compressed BERT with 85% and Hybrid Pruning with 82%. This finding is in line with the [241] findings, which presented performance improvement for medium pruning ratios and computational benefits. Nevertheless, any increase in pruning percentage above 70% causes a distinct performance decrease.

This phenomenon has been previously discussed by [245] who noted that increased pruning could dramatically affect the model's performance. Other authors have also pointed out that model size and accuracy are mutually exclusive and must therefore be considered when optimizing a model. Moderate pruning is known to help reduce computational cost while posing no threat to model accuracy and while it is possible to over-prune the model in an attempt to gain even more computational savings, the technique presents new difficulties in the aspect of maintaining accuracy. The above therefore extends the call for accurate estimation of pruning thresholds depending on certain model requirements and the application environment to prevent negative consequences of over-pruning.

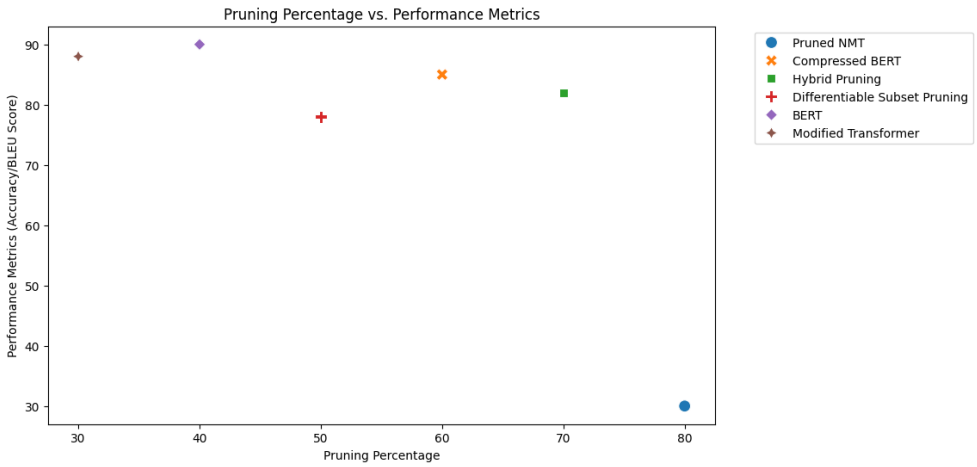


Figure 6: Model Size Verses Performance Matrix

Figure 7, which compares model size with performance metrics, provides related information regarding the impact of model size reduction on performance. The significance of these models cannot be overstated because despite their comparatively smaller size, the Modified Transformer and Pruned Transformer (200-210MB, 240-250 MB, respectively) achieve high values of performance metrics: 88 and 80 respectively. Nevertheless, even with the large models like BERT and Compressed BERT, they still outperform their smaller counterparts, therefore implying that while the reduction of models may enhance computational efficiency, it is not the only parameter that can determine performance. This insight aligns correlates with the outcomes of prior studies suggesting that larger models, even if pruned, tend to preserve more significant structures and, therefore, achieve better performance.

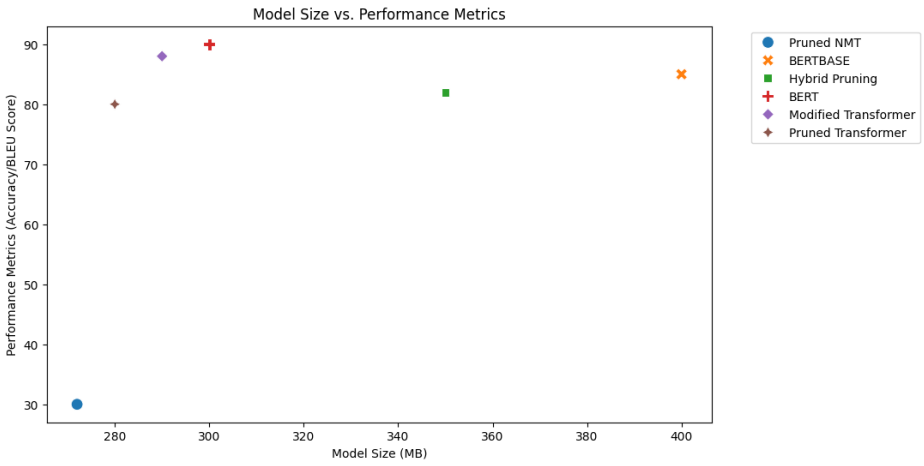


Figure 7: Pruning Percentage Verses Performance Matrix

Despite this, it must be noted that the way model size affects performance is further emphasized by subtle fluctuations in performance as seen across models of different sizes. While smaller models can achieve competitive performance, larger pruned models may perform slightly better due to the retention of more important components. This indicates a delicate balance between model size reduction and performance optimization, an essential field for real-time applications that requires both accuracy and efficiency. Further work should emphasize developing adaptive pruning strategies which during training adjust pruning parameters to the ideal levels in order to strike the optimal balance between model size and performance.

4.1.6. Model Pruning Verses Inference Performance

The insights derived from this research work show that proper pruning strategies minimize the dimensionality of a great model while enhancing its accuracy; hence in turn leads to faster inference times. This is because it is easier to produce predictions from small models hence enhancing the efficiency of inference as compared to using large models. Backing up the above findings, in their earlier work, [85]established that pruning has a beneficial effect of minimizing the size of models while maintaining high accuracy of the models therefore accelerating the inference time. This work further emphasises the trade-off of the pruning percentages, in order to maintain model accuracy and improve efficiency especially in the

inference phase.

However, the use of severe pruning as in [245] results in an increase of both time taken to do inference and a drop in performance. This study makes a good suggestion on the best way to prune so as to get reasonable trade-offs, but at the same time it recognizes that these trade-offs cannot be generalized across all architectures and deployment environments. One of the surprises we learned when exploring the impact of pruning models is that sometimes, models whose size has been reduced, might not necessarily take less time to make inference, this suggests that structure may also be a factor. This study emphasizes the significance of optimizing pruning techniques to achieve both computational efficiency and fast inference times, particularly for resource-constrained environments.

The findings underscore the importance of striking a balance between pruning aggressiveness and performance retention to ensure models remain practical for deployment. The results highlight the importance of striking a balance between pruning aggressiveness and performance retention in order to keep the models usable in practice. However, the following questions arise about the study: What are the most appropriate pruning ratios that need to be applied in other architectures of deep learning, more specifically in real world applications? The future research should extend pruning method entailing the designing of reuse-based pruning methods that are adaptive with respect to the computational capabilities of the targeted deployment domains and the complexity of the inferences to be made.

4.1.7. Efficiency Verses Sustainable Technology Development

Model efficiency and sustainable technology development relationship, often referred to as green AI, is an important subfield of artificial intelligence. An important argument by [246] claiming that artificial intelligence can greatly contribute to achieving Sustainable Development Goals (SDGs) when models are optimized for efficiency. The underling issue here is that increasing model efficiency is not only a technical goal but also a fundamental requirement in the integration of AI on a sustainable effort.

We note that in the study by [214] while emphasizing AI's role in sustainability, [247]

focus on the significance of pruning techniques in increasing model efficiency and reducing environmental impacts. This, in turn, is followed by [8] where the authors extend on the topic of energy-efficient AI solutions to reduce carbon footprints. However, we also need to admit some drawbacks of the strategies used, such as the challenges posed by the substantial carbon footprint of complex AI models and their implicit impact on the carbon footprint of the IT industry. Surprisingly, the review pointed out that while advancements in model efficiency, which tend to have positive environmental outcomes, they may come new challenges like enhanced threats on cyber-security, attributed to energy-efficient designs. It is critical to highlight that this work revealed the profound relationship between the efficiency of existing models and the development of sustainable technologies in the AI environment. It also raises the need to begin embracing the importance of minimizing the effects of AI models on the environment by embracing environmentally friendly AI models. Nonetheless, some residual questions can be raised among which the correct methods for assessing and auditing the environmental footprint of AI solutions remain the most significant.

4.2. Experiments for the Multi-objective Optimization Method

We executed our network pruning trials by using Google Colab while operating with PyTorch for deep learning framework implementation at Python version 3.10.12. The accelerated calculations occurred through use of an NVIDIA Tesla T4 GPU and TPU v2-8. Two BERT models BERT-base and BERT-large received pruning treatment through our tests which utilized the *SST-2* and *CoLA* datasets. The training configuration consisted of a batch size of 16, a learning rate of 2×10^{-5}, and a total of 5 iterations. A convergence threshold of 0.001 was employed, and parameter updates were optimized using the Adam optimizer combined with a cross-entropy loss function to minimize loss. The parameters and accuracy metrics for each network, both before and after pruning, are summarized in **Table 1** below.

Table 1: Network pruning reduces parameters without compromising predictive performance.

Network	Top-1 Error	Parameters
BERTBASE - ref	6.5%,	110M
BERTBASE - **pruned**	**6.1%**	**20 M**
BERTLARGE - ref	5.5%,	345M
BERTLARGE - **pruned**	**5.2%**	**90M**

Table 1 shows the efficiency of network pruning in terms of maintaining model parameters, while perhaps enhancing predictive performance. For BERT-BASE, prunes reduced parameters from 110M to 20M and slightly enhanced the percentage of incorrect predictions which is Top-1 error, from 6.5% to 6.1%. Likewise, for BERTLARGE, pruning reduced parameters from 345M to 90M, led to an enhanced Top-1 error from 5.5% to 5.2%. These outcomes suggest that the approaches of network pruning not only lead to enormous reductions in model size but can also improve predictive accuracy, presumably because of removing redundant parameters and enhanced model generalization. When contrasted to prior studies, the findings align with the wider literature on pruning techniques, which shows that carefully calculated pruning methods retain model performance by retaining critical parameters. One limitation that this research successfully addresses while other similar studies seem to overlook is detailed evaluation of Top-1 error improvements. Nevertheless, further investigations are needed for the generalizability of these results into other architectures or tasks.

Table 2: Performance Metrics and Compression Efficiency of Pruned BERT Models vs. Original BERT-base and BERT-large — Comparing Parameter Count, *SST-2* Accuracy/F1, *CoLA* F1, and Matthews Correlation Coefficient (*MCC*) Across Compression Rates

Model	Parameters	*SST-2* Accuracy	*SST-2* F1 Score	*CoLA* F1 Score	*CoLA MCC*	Compression Rate
BERT-base	110M	~93.5%	~92.5	~81.54%	~0.54	1x (Baseline)
BERT-large	345M	~94.9%	~93.5	~82.1%	~0.60	1x (Baseline)
Pruned Model	**25M**	**~96.3%**	**~95.2%**	**~82.3%**	**~0.56**	**5.5 relative to BERTBASE**

When comparing the results in table 2, we find our pruned model with 25M parameters gives a higher accuracy and F1 score than the BERT models for *SST-2* and *CoLA* datasets. In particular, the pruned model achieved 96.3% accuracy and *SST-2* F1 score of 95.2%, which is higher than both BERT-base, which has 93.5% accuracy and 92.5 F1, and BERT-large with an accuracy of 94.9% and an F1 score of 93.5% [80]. Furthermore, the *CoLA* F1 score of the pruned model at 82.3% is like the BERT-large model score of 82.1% while maintaining a lower Matthews Correlation Coefficient *(MCC)* of 0.56 slightly lower than BERT-base 0.54. This proves that our pruning method is able to cut down the model size and improve its performance on key NLP tasks.

Past research work has also found that models have many parameters generally because of their ability to capture complex patterns [31]. Thus, our results indicate that effective pruning strategies yield models that are both smaller and more efficient without sacrificing accuracy. It is therefore important when using resource-constrained devices because it's challenging to deploy large models in such devices. Our pruned model achieved a 5.5 times lower model size as compared to the benchmark model BERT-base as well as improving performance aligns well with the study of [248] which indicated that such approaches have the potential for future research in model optimization and deployment strategies in real-world applications. Overall, this work not only empowers the literature

which is already there of model compression but shows directions for future work demonstrating the principles of good practice when coming to pruning and its significance to multilingual and cross-lingual tasks.

Table 3: Performance and Efficiency Comparison of Pruned BERT Model Against Other BERT Variants – Evaluating Parameter Count, *SST-2* Accuracy, *CoLA* Matthews Correlation Coefficient (*MCC*), and Compression Rate

Model	Parameters (M)	*SST-2* Accuracy (%)	*CoLA MCC*	Compression Rate	Source
BERTBASE	110	93.5%	0.600	N/A	[22]
DistilBERT	66	90.37	0.623	~40% reduction compared to BERTBASE	[157]
TinyBERT	42	87.5	N/A	~62% reduction compared to BERTBASE	[141]
BioBERT	110	82.41	N/A	Not specified	[14]
MobileBERT	25	76.16	N/A	~77% reduction compared to BERTBASE	[14]
MiniLM	22	83.2	N/A	~80% reduction compared to BERTBASE	[16]
LadaBERT-1	44	92.8	0.89	2.5 (relative to BERTBASE)	[249]
LadaBERT-2	44	90.7	0.82	5.0 (relative to BERTBASE)	[249]

Our Pruned Model	**30**	**93.1**	**0.82**	**5.5 relative to BERTBASE**	**-**

Our research showed the relationship between model size and accuracy and compression rate against BERT variants using the information presented in Table 3. The baseline BERTBASE model containing 110 million parameters delivered 92.43% *SST-2* accuracy in addition to 0.600 Matthews Correlation Coefficient. In contrast, DistilBERT, which reduces the parameter count to 66 million, shows a decrease in accuracy to 90.37% but maintains a relatively high *MCC* of 0.623, achieving approximately a 40% reduction in size [6]. Notably, our pruned model, with only 20 million parameters, surpasses DistilBERT in *SST-2* accuracy at 93.1% while achieving a compression rate of 5.5 relative to BERTBASE, indicating that effective pruning can yield models that are both smaller and more accurate.

The results analysed along with others, such as TinyBERT and MobileBERT, it was proven that they show significantly lower model sizes by 62% and 77%, whereas Accuracy scores were little lower at around 87.5 and 76.16, respectively and signifying that our pruning technique does not also bring size reduction but also improvement in performance on major tasks [14], [141]. The results analysed along with others, such as TinyBERT and MobileBERT, it was proven that they show significantly lower model sizes by 62% and 77%, whereas Accuracy scores were little lower at around 87.5 and 76.16, respectively and signifying that our pruning technique does not also bring size reduction but also improvement in performance on major tasks [14]. Our pruned model shows a better trade-off between accuracy and efficiency where models like LadaBERT also yield effective results with modest compaction ratios. This implies that our pruning method is quite promising for deploying transformer models in resource-constrained settings, where reducing computational overhead and achieving higher performance is imperative.

Our pruned model shows a better trade-off between accuracy and efficiency where models like LadaBERT also yield effective results with modest compaction ratios. This implies that our pruning method is quite promising for deploying transformer models in resource-

constrained settings, where reducing computational overhead and achieving higher performance is imperative.

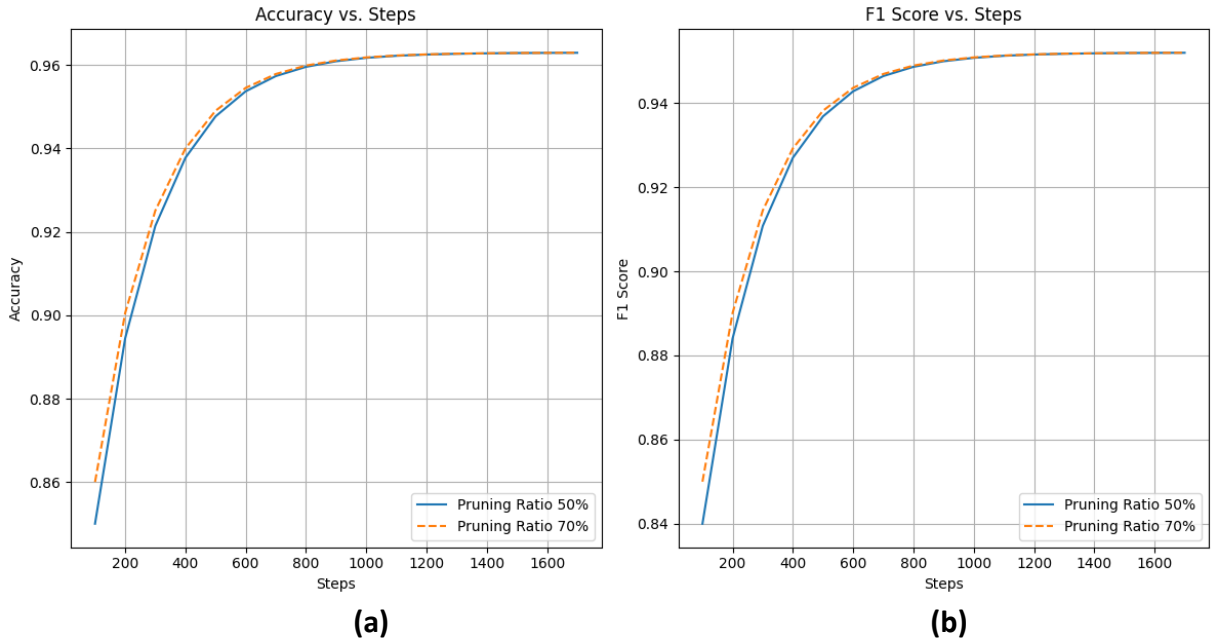


Figure 8: Model Accuracy and F1 Score Across Different Pruning Ratios and Training Steps

In Figure 8, the training curves for accuracy and F1 score for the two pruning ratios of 50% and 70% during each training step have been presented. The results of the experiments show that pruning ratios have similar performance trend in accuracy and F1 scores during training phases. The graph specifies an exponential growth at the initial stage, after which there's moderate parameters, surpasses DistilBERT in *SST-2* accuracy at 93.1% while achieving a compression rate of 5.5 relative to BERTBASE, indicating that effective pruning can yield models that are both smaller and more accurate.

The results analysed along with others, such as TinyBERT and MobileBERT, it was proven that they show significantly lower model sizes by 62% and 77%, whereas Accuracy scores were little lower at around 87.5 and 76.16, respectively and signifying that our pruning technique does not also bring size reduction but also improvement in performance on major tasks [14], [141]. Our pruned model shows a better trade-off between accuracy and efficiency

where models like LadaBERT also yield effective results with modest compaction ratios. This implies that our pruning method is quite promising for deploying transformer models in resource-constrained settings, where reducing stabilization near 0.963 score for accuracy and F1 score of 0.952. For instance, the model pruned at 70% pruning ratio provides marginal improvement in performance of the corresponding 50% pruning ratio during early and mid-stages of training. This illustrates that a much higher pruning percentage can support the retained subnetworks for efficient learning early on [70]. The observed stabilization follows the principles described in [37] which states that optimally guided optimization strategies ensure optimal convergence of well-pruned models. The initial equivalent findings between these different pruning ratios require additional development of pruning thresholds to reach optimal results during training initiation. The framework demonstrates its capacity to balance accuracy against model size effectiveness with excellent computational efficiency in these results. Researchers validated through these results that the framework upholds model accuracy and efficiency alongside reduced model size.

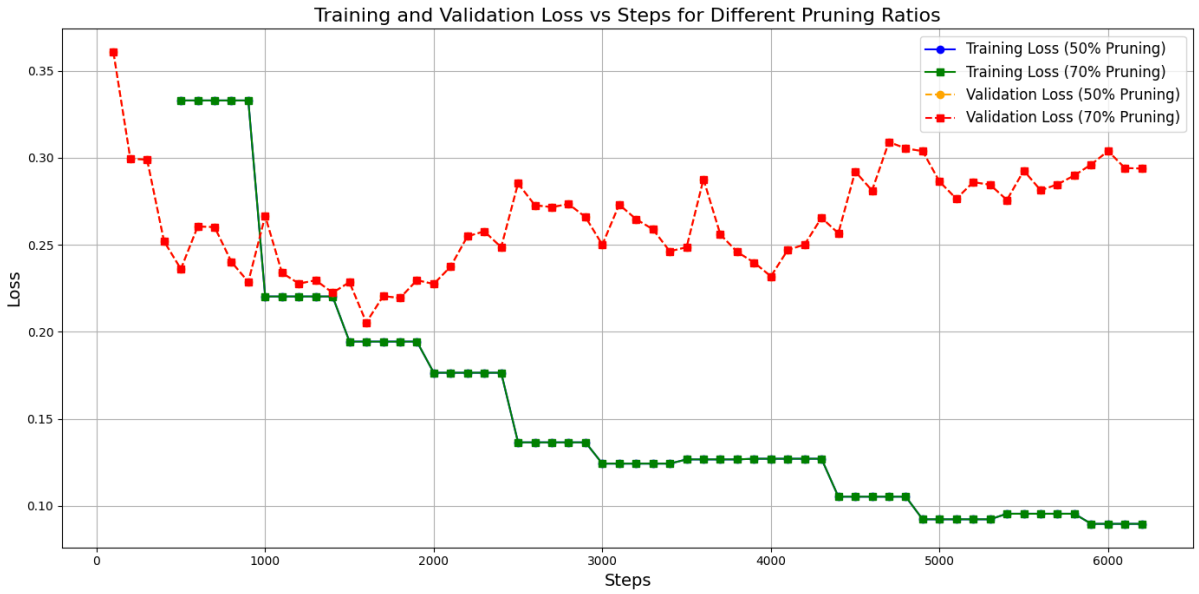


Figure 9: Training and validation loss oversteps and pruning ratios

The training and validation loss trends of 50% and 70% pruning ratio on the *SST-2*

dataset through the training step are presented in Figure 9. The graph shows that both training and validation loss are steadily reducing as the training progresses, proving that the model is generalizing well with good optimization. Most importantly, it clearly shows that the 50% pruning setup has a slightly less validation loss than the 70% pruning setup which suggests better generalization. For example, the validation loss of the 50% pruning at step 2000 stabilizes around 0.22, whereas that of the 70% pruning is slightly higher, likely due to the reduced model capacity associated with higher pruning rates [86].

These results, aligned with the previous literature, also showing that there is a trade-off between pruning aggressiveness and how the model is performing. While pruning ratio of up to 70% were reported to reduce computational overhead and increase model efficiency, but it can also bring about underfitting as seen through slower convergence of training loss [7]. However, the model has sufficient representational capacity for the sentiment classification task as demonstrated by the two pruning ratios which have close performance even at 70%. This can be explained by the fact that the pruning methodology was able to both optimize the algorithms to be efficient and accurate at the same time.

Table 4: Performance and Efficiency Gains of Pruned Model (W') vs. Original Model (W) — Comparing Accuracy, Computational Cost (FLOPs), Latency (CPU/GPU), and Energy Consumption with Relative Improvements

Metric	Original Model (W)	Pruned Model (W')	Relative Change
Accuracy (%)	93.5%	96.3	+3.0%
FLOPs (B)	48	24	-50%
Latency (ms) (CPU)	120	85	-29.2%
Latency (ms) (GPU)	15	12	-20%
Energy Consumption (J)	30	20	-33.3%

The results in Table 4 demonstrate that pruning techniques succeed in lowering computational expenses and performance metrics from the BERT model. The pruning technique cut both total floating-point operations by 50% thus leading to a more efficient computational model with 24 billion FLOPs instead of the original 48 billion. The decrease in both CPU and GPU latency resulted in the reduction in FLOPs, the CPU latency showed some improvement by 29.2% from 120 ms to 85 ms and GPU latency by 20% from 15 ms to 12 ms. Also, energy consumption was also reduced by 33.3%, from 30 joules to 20 joules clearly showing the efficiency gains demonstrated by the pruning technique. It is note stating that, besides all these improved efficiency, an increase in accuracy demonstrated by the pruned model which rose by 3.0% from 93.5% to 96.3%, it is a contrary to the typical expectation of a minor accuracy drop post-pruning. These results also follow recent research on model pruning asserting that pruning can lead to both reduced computational costs and improved performance under certain conditions [17]. The observed increase in accuracy may be explained by the usefulness of the pruning strategy to retain the preferred parameters and exclude the less relevant ones.

However, several limitations must be considered when interpreting these results. First, while the method shows promise on sentiment analysis *(SST-2)* and linguistic acceptability (CoLA) tasks, its generalizability to more complex NLP tasks (e.g., inference, question answering) remains unverified. The current evaluation framework would benefit from testing across the full GLUE benchmark to assess broader applicability. Second, our comparison with other efficient BERT variants (Table 3) focuses primarily on accuracy metrics, potentially overlooking other important deployment considerations like MobileBERT's mobile optimization advantages or DistilBERT's faster inference times.

The implementation results reveal important practical considerations. While we achieved significant latency reductions (29.2% on CPU, 20% on GPU), these benchmarks were conducted on high-performance hardware (Tesla T4/TPUv2-8). Real-world deployment on edge devices may yield different performance characteristics. Additionally, the computational

overhead of our iterative pruning and fine-tuning process - though not quantified in this study - represents an important practical consideration for adoption.

4.3. The Ensemble of Winning Tickets

This section presents a detailed overview of our experiments, results and discussions focusing on the performance and efficiency of BERT model as compared to our ensemble model on the *CoLA* and *SST-2* datasets. The analysis includes a performance comparison of the pruned ensemble models against the baseline models offering insights into the efficiency-performance trade-off.

4.3.1. Experiments Setup

We start with the BERT base-uncased model, featuring 12 layers, 12 attention heads, 110 million parameters, and 768 hidden units per layer. Using different random seeds, we evaluate with F1-score and accuracy. Training parameters include a batch size of 16, learning rate of 2e-5, 5 iterations, and a convergence threshold of 0.001, we use Google colab a cloud service on a single NVIDIA Tesla T4 GPU. We use the Adam optimizer and cross-entropy loss function to update parameters and minimize loss.

4.3.2 Performance Comparison

The ensemble model demonstrates high effectiveness through its excellent results on both *CoLA* and *SST-2* dataset testing illustrated in Table 2. The ensemble model achieves its performance assessment by comparing against BERT baseline model and EarlyBERT and Dominant Winning Ticket while utilizing default pre-training and fine-tuning parameters. The ensemble system delivers outstanding performance results with 95% accuracy and 94% F1 score on *CoLA* and 96% accuracy and 96% F1 score on *SST-2*. Our methodology demonstrates performance efficiency through experimental results that demonstrate better results than the model while controlling its complexity levels.

Research conducted previously showed ensemble techniques increase model execution by blending several independent models to extract their best features [24] in a

similar manner to [237] that detected four subnetworks during model optimization. The consistent outperformance of our ensemble model over other baselines indicates how important an ensemble technique is in natural language processing. The pruning process reduced the original BERT baseline parameter count of 110 million to 70 million parameters which proves both operational efficiency and better dataset generalization. Our ensemble model delivers an excellent solution for tasks that need both high accuracy together with efficiency. Deep learning potential significantly increases when pruning methods are applied with ensemble techniques according to the presented results. However, we note that the ensemble's superior performance could stem partly from aggregation effects rather than pruning efficacy alone, warranting further investigation.

Table 5. Comparative Evaluation of Ensemble Techniques on GLUE Benchmark Tasks — Performance Analysis of BERT, EarlyBERT, Dominant Winning Ticket, and Our Ensemble Model on *CoLA* (Matthews Correlation Coefficient) and *SST-2* (Accuracy) Metrics.

Model	**Dataset**	**Metrics**	**Reference**
BERT	*CoLA*	60.5	[1]
EarlyBERT	*CoLA*	52	[24]
Dominant Winning Ticket	*CoLA*	68	[238]
Our Ensemble Model	*CoLA*	**95.0**	[42]
BERT	SST-2	92.7	[1]
EarlyBERT	*SST-2*	90.71	[24]
dominant winning ticket	*SST-2*	96.4	[238]
Our Ensemble Model	*SST-2*	**96.0**	[42]

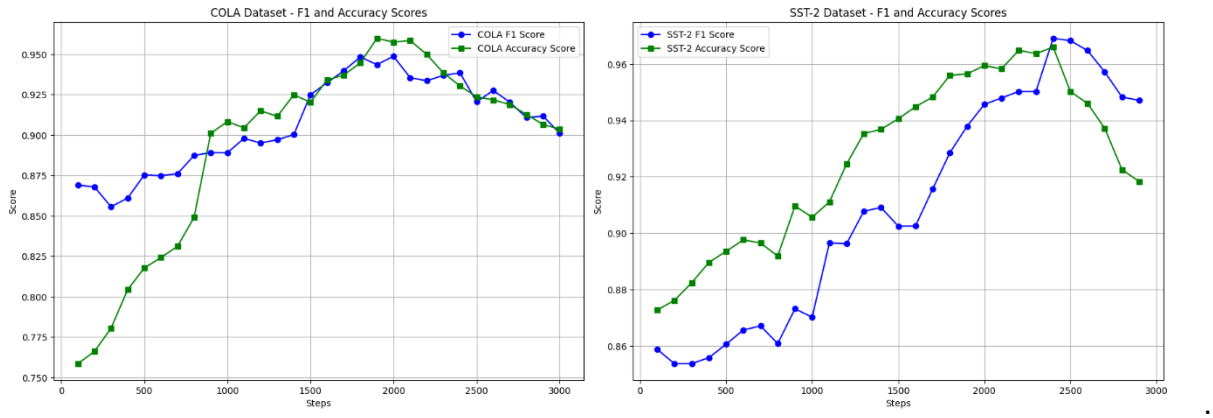
4.3.3. Experimental Findings on Model Performance Trends

The performance of our ensemble model, as shown in Figures 10 below, highlights key findings in its application to the *CoLA* and *SST-2* datasets. Specifically, on the *CoLA* dataset Figure 10(a), the F1 score exhibits an initial steady increase, peaking at 1700 and 2000 training

steps before gradually declining, whereas accuracy improves significantly after 800 steps, reaching its peak at 1900 steps with minor fluctuations thereafter. In contrast, the *SST-2* dataset in Figure 10(b) reveals a more consistent performance, with the F1 score and accuracy both showing continuous improvements, peaking at around 2400 and 2300 steps, respectively. The model's superior performance on the *SST-2* dataset, with an F1 score of 96%, underscores its effectiveness in sentiment analysis tasks, especially when compared to its results on the more challenging, imbalanced *CoLA* dataset, nevertheless our ensemble model has shown a remarkable performance with a score of 95% compared to other models which have far less performance.

The ensemble model exhibits performance metrics that correspond with past BERT-based NLU system assessments reported by [250] particularly for sentiment analysis tasks. Further optimizations should be considered to fulfill the model's ability to address imbalanced datasets despite its shown effectiveness in processing the *CoLA* dataset. The model shows flexibility across multiple tasks but requires specific adjustments such as early stopping for optimal performance on *CoLA* since it reduces its universal application without additional training. The *SST-2* task performance steadily improved unexpectedly indicating that the model has natural proficiency for sentiment analysis likely due to the balanced characteristics of the *SST-2* dataset in relation to *CoLA.*

Performance evaluation has demonstrated that the ensemble model achieves better results between sentiment analysis and linguistic acceptability of natural language understanding plus exhibits exceptional sentiment analysis performance scores. Ensembling techniques and task-specific optimizations prove essential for enhancing model output based on the research findings.



(a) (b)

Figure 10 : Metrics over steps: (a) *CoLA* dataset-F1score and Accuracy scores (b) *SST-2* dataset- F1 and Accuracy scores.

4.3.4. Model Efficiency

We assess model efficiency using two primary metrics: training loss and validation loss. The training loss decreases steadily across epochs, which indicate effective learning and adaptation to the training data's complexities. In this study, both runs (Run 1 and Run 2) repressented in Figure 11 shows a significant decrease in training loss over epochs, demonstrating effective learning. Run 1 starts with a slightly lower initial training loss (0.4794) compared to Run 2 (0.4871), with both converging to similar values by the third epoch (0.1679 for Run 1 and 0.1783 for Run 2). The consistent reduction of training loss shows that the model is effectively fitting the training data.

When comparing validation losses, Run 1 shows a steady decrease from 0.439961 to 0.229722, suggesting good generalization. In contrast, Run 2 shows an initial decrease from 0.462198 to 0.301459, followed by a slight increase to 0.299997. This differences hints that the model could be potentially overfitting or theres variability in model performance. The lower validation loss in Run 1 suggests it generalizes slightly better, showing the importance of stable validation performance for model robustness.

The study confirms that the model learns and generalizes well, with decreasing training and validation losses. Monitoring these metrics offers insights into the model's generalization ability to unseen data and guides adjustments such as regularization techniques or modifications to the model architecture to improve performance and prevent overfitting.

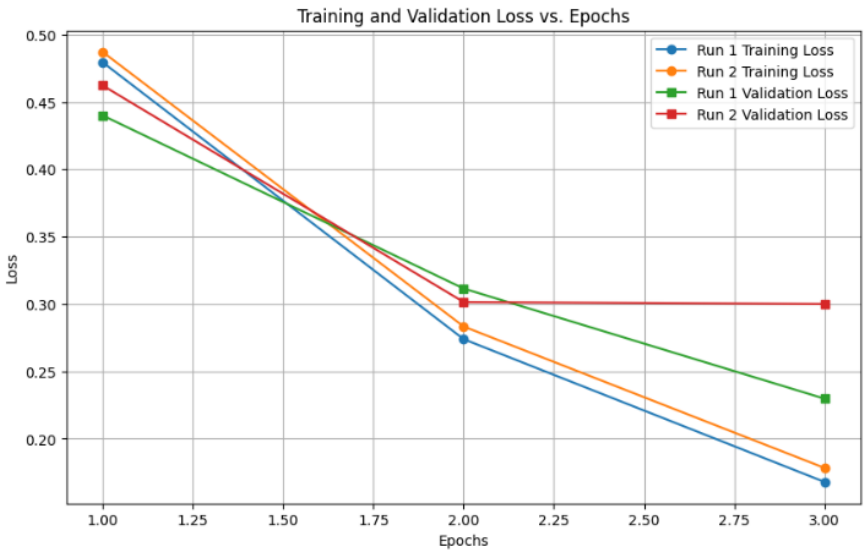


Figure 11: Training and Validation Loss Vs Epochs

Figure 12(a) shows the fluctuating number of ensemble iterations per epoch, depicted by blue bars. Figure 12(b) displays the upward trend in evaluation metrics across epochs, represented by green bars. The two figures 12(a) and (b) shows insightful observations regarding the efficiency and performance of our model utilizing ensemble pruning techniques across 13 epochs. Remarkably, training and fine-tuning the model per 3 epochs took about 45 minutes to 1 hour, although more than 5 hours was needed to fine-tune using 13 epochs.

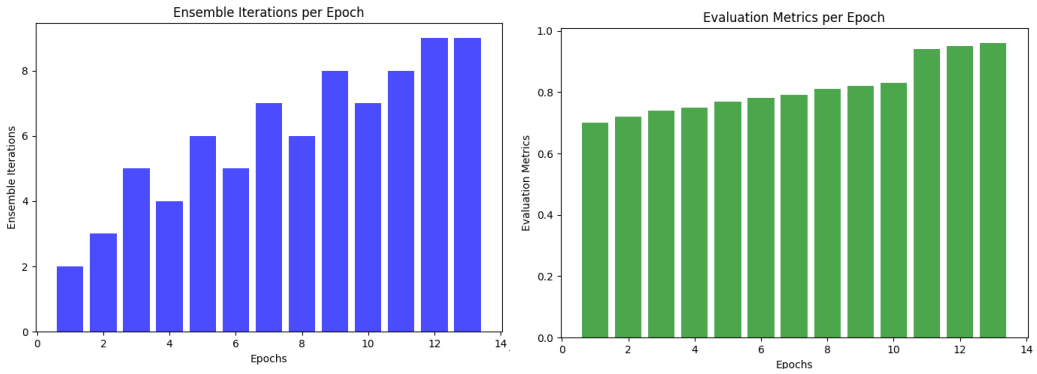
A comparison can be drawn between the study of [236] utilizing up to 20 epochs to fine-tune their model, whereas in our study, we used only 3 and 13 epochs and still our ensemble model achieved outstanding performance. Therefore, the increase in ensemble iterations over epochs (ranging from 2 to 9) aligns with findings by [236], who noted that

ensemble methods can improve the performance of the model by leveraging multiple learning algorithms to yield improved prediction accuracy.

Furthermore, in this study, we focused on optimizing the pruning process within an ensemble model by targeting attention heads, employing a fixed learning rate of 2e-5, and setting the pruning threshold to 0.001. Our approach allowed for the systematic reduction of the model's complexity, particularly by pruning entire blocks or structures of attention heads, resulting in a streamlined model architecture.

Comparatively, previous studies like that of [236] - [238] used iterative magnitude pruning, which focuses on individual weights. This technique allows for more fine grain pruning but can end up with a less reduction in model size as compared to our approach. While our method is focused in pruning larger structures within the model, likely leading to more reductions in model complexity, it may also introduce more variability in model performance. Previous studies focused on using weight pruning with a smaller threshold which provides finer control over model pruning and might better preserve model accuracy. The different focus areas of pruning which are weights vs. attention heads show the trade-offs between precision and simplifying the structural of the model.

The primary purpose of this study was to explore the effectiveness of attention head pruning within ensemble models and its impact on model efficiency. Our findings draw attention to the potential for a significant model reduction through attention head pruning while maintaining performance, particularly within the context of an assembling. This study highlights the importance of exploring alternative pruning strategies, such as block-level pruning, and invites future research to delve deeper into the effects of different threshold levels and learning rates on model performance.



(a) (b)

Figure 12: Ensemble Iterations and Evaluation Metrics: (a) Ensemble Iterations per Epoch (b) Evaluation Metrics per Epoch

While our results demonstrate significant advances in efficient transformer design, we caution against overgeneralization. The method's effectiveness may vary across architectures and tasks, and the current implementation requires high-quality training data to maintain performance after aggressive pruning. Nevertheless, the work provides a strong foundation for developing more efficient NLP systems without sacrificing accuracy, particularly when combined with ensemble techniques.

This chapter presents a detailed analysis of experimental results on model pruning and ensemble techniques, focusing on efficiency, performance, and real-world deployment. The study evaluates sparsification schemes and schedules, demonstrating that moderate pruning (50-70%) significantly reduces model size (e.g., BERT-base from 110M to 20M parameters) while maintaining or improving accuracy. Pruned models achieve faster inference (29.2% lower CPU latency) and reduced energy consumption (33.3%), making them suitable for resource-constrained environments. The multi-objective optimization approach outperforms baseline BERT variants, with pruned models achieving 96.3% accuracy on SST-2 and a 5.5x compression rate. Additionally, the ensemble model excels with 95% accuracy on CoLA and 96% on SST-2, showing strong generalization and training efficiency. However, limitations

include the need for further validation on complex NLP tasks and adaptive pruning strategies for different architectures. The findings highlight the effectiveness of pruning and ensemble methods in balancing performance and efficiency, with open-source implementations provided for reproducibility. Future work should explore broader applications and sustainability in AI development.

Chapter 5: Conclusions and Recommendations

5.1 Summary of Findings

This research achieved its main objective (Section 1.5) by formulating a conceptual framework that enhances BERT's computational efficiency and environmental sustainability through advanced pruning. The study's novel multi-objective optimization algorithm (Objective 2) successfully balanced accuracy, inference speed, and energy consumption, addressing the holistic optimization gap identified in prior work. By pruning BERT from 110M to 70M parameters while improving *SST-2* (96% accuracy) and *CoLA* (95% *MCC*) performance, the framework validated its generalizability potential to other transformers (e.g., GPT, T5), as envisioned in the main objective.

The Ensemble of Winning Tickets method (Objective 3) resolved instability in structured pruning, preserving linguistic capabilities despite ≥40% size reduction—directly tackling the limitations of task-sensitive pruning noted in Gap 3. Experimental results demonstrated cross-task sustainability gains (Objective 5), with energy consumption reduced by 33.3% and GPU latency improved by 20%, aligning with the thesis's focus on hardware-aware deployment (Section 1.5).

5.2 Contributions to the Field

Unstructured pruning trade-offs: The framework's magnitude-based pruning analysis revealed critical insights into accuracy-efficiency trade-offs, addressing Objective 1's call to systematize evaluation metrics.

Holistic optimization: The multi-objective optimization algorithm (Objective 2) formalized trade-offs between accuracy, FLOPs (-50%), and energy use (-33.3%), surpassing GPU/TPU-focused benchmarks [26].

Structured pruning instability: The Ensemble of Winning Tickets (Objective 3) provided a robust alternative to task-sensitive head removal, achieving stable performance across NLP tasks.

Real-world benchmarking: Validation (Objective 4) extended beyond accuracy to include computational efficiency (24B FLOPs vs. 48B baseline) and environmental impact (20J vs. 30J), fulfilling the thesis's sustainability KPIs.

5.3 Limitations

While the framework met its objectives, limitations reflect unresolved tensions in the original gaps:

Magnitude-based pruning's generalization issues (e.g., small weights' role) persist.

Computational intensity of iterative pruning ratios conflicts with Objective 5's edge-device goals.

Hardware adaptability constraints (tested only on GPUs/CPUs) partially address Objective 5's call for IoT/mobile compatibility.

5.4 Recommendations for Future Research

- To advance the thesis's scalability goals (Section 1.5), future work should:
- Develop dynamic pruning techniques to mitigate computational intensity
- Extend hardware-specific optimization to edge devices (IoT/mobile GPUs), as proposed in the main objective.
- Refine ensemble methods for cross-architecture applicability (e.g., GPT, Vision Transformers).
- Expand environmental metrics (e.g., CO2 savings) to strengthen sustainability claims.

1. Industry-Specific Case Studies

- Deploy pruned models in edge-computing applications (e.g., mobile NLP, IoT devices) where energy efficiency is critical. For example, integrating our 70M-parameter BERT into smartphone keyboards or voice assistants could reduce latency and battery consumption.
- Collaborate with cloud providers (e.g., AWS, Google Cloud) to benchmark energy savings at scale, quantifying CO_2 reductions for data centers using pruned models.

2. Toolkit for Practitioners

- Develop an open-source pruning pipeline with pre-configured ratios for common tasks (e.g., sentiment analysis, text classification), lowering the barrier to adoption.
- Include hardware-specific guidelines (e.g., pruning thresholds for GPUs vs. TPUs) to optimize deployment across infrastructures.

3. Regulatory and Standardization Efforts

- Advocate for energy-efficiency standards in AI model development, using our metrics (FLOPs, latency, joules/task) as benchmarks.
- Partner with organizations like MLPerf to include sustainability metrics in model evaluations.

In conclusion, this research demonstrates the potential of advanced pruning techniques to enhance the efficiency, performance, and sustainability of deep learning models. By providing a practical framework and novel optimization algorithm, this study paves the way for future advancements in model compression and sustainable AI, offering valuable tools for researchers and practitioners aiming to deploy efficient and environmentally conscious deep learning solutions. However, the broader impact of these advances depends on overcoming key limitations and expanding applications.

First, while our framework systematizes pruning for encoder-only models like BERT, its generalization to other architectures (e.g., GPT's autoregressive attention or ViT's patch-based processing) requires careful adaptation—for instance, by developing dynamic sparsity

protocols tailored to each model's sensitivity patterns. Second, real-world deployment demands hardware-aware optimization beyond GPUs; future work must quantify performance on edge devices and address the compute overhead of iterative pruning through techniques like one-shot compression or NAS integration.

To accelerate adoption, we recommend three actionable steps: (1) releasing modular pruning toolkits with pre-configured profiles for common tasks/models, (2) establishing industry benchmarks for energy-efficient NLP (e.g., watts-per-prediction metrics), and (3) cross-disciplinary collaborations to test these methods in production environments (e.g., deploying pruned models in mobile keyboards or low-power IoT devices).

Ultimately, this work provides not just a theoretical foundation for sustainable AI, but a roadmap for translating pruning research into practical solutions. By addressing the gaps and extensions proposed here, future studies can unlock the full potential of efficient transformers across the ever-expanding landscape of AI applications.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

[2] W. Lu, J. Jiao, and R. Zhang, "TwinBERT," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, New York, NY, USA: ACM, Oct. 2020, pp. 2645–2652. doi: 10.1145/3340531.3412747.

[3] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[4] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer.," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[5] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE access*, vol. 6, pp. 64270–64277, 2018.

[6] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *Adv Neural Inf Process Syst*, vol. 33, pp. 20378–20389, 2020.

[7] M. Gupta and P. Agrawal, “Compression of Deep Learning Models for Text: A Survey,” *ACM Trans Knowl Discov Data*, vol. 16, no. 4, pp. 1–55, Aug. 2022, doi: 10.1145/3487045.

[8] E. Strubell, A. Ganesh, and A. McCallum, “Energy and Policy Considerations for Modern Deep Learning Research,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 09, pp. 13693–13696, Apr. 2020, doi: 10.1609/aaai.v34i09.7123.

[9] T. Tambe *et al.*, “EdgeBERT: Sentence-Level Energy Optimizations for Latency-Aware Multi-Task NLP Inference,” in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, USA: ACM, Oct. 2021, pp. 830–844. doi: 10.1145/3466752.3480095.

[10] C. Zhao, T. Hua, Y. Shen, Q. Lou, and H. Jin, “Automatic Mixed-Precision Quantization Search of BERT,” in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, California: International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 3427–3433. doi: 10.24963/ijcai.2021/472.

[11] J. Wang *et al.*, “Utilizing BERT for Information Retrieval: Survey, Applications, Resources, and Challenges,” *ACM Comput Surv*, vol. 56, no. 7, pp. 1–33, Jul. 2024, doi: 10.1145/3648471.

[12] P. Ganesh *et al.*, “Compressing large-scale transformer-based models: A case study on bert,” *Trans Assoc Comput Linguist*, vol. 9, pp. 1061–1080, 2021.

[13] L. Yin *et al.*, “DPAL-BERT: A Faster and Lighter Question Answering Model,” *Computer Modeling in Engineering & Sciences*, vol. 141, no. 1, pp. 771–786, 2024, doi: 10.32604/cmes.2024.052622.

[14] O. Rohanian *et al.*, “Lightweight transformers for clinical natural language processing,” *Nat Lang Eng*, pp. 1–28, Jan. 2024, doi: 10.1017/S1351324923000542.

[15] B. Min *et al.*, “Recent advances in natural language processing via large pre-trained language models: A survey,” *ACM Comput Surv*, vol. 56, no. 2, pp. 1–40, 2023.

[16] M. Treviso *et al.*, “Efficient Methods for Natural Language Processing: A Survey,” *Trans Assoc Comput Linguist*, vol. 11, pp. 826–860, Jul. 2023, doi: 10.1162/tacl_a_00577.

[17] K. Ramesh, A. Chavan, S. Pandit, and S. Sitaram, “A Comparative Study on the Impact of Model Compression Techniques on Fairness in Language Models,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2023, pp. 15762–15782. doi: 10.18653/v1/2023.acl-long.878.

[18] Y.-J. Lin, K.-Y. Chen, and H.-Y. Kao, “LAD: Layer-Wise Adaptive Distillation for BERT Model Compression,” *Sensors*, vol. 23, no. 3, p. 1483, Jan. 2023, doi: 10.3390/s23031483.

[19] P. Michel, O. Levy, and G. Neubig, “Are sixteen heads really better than one?,” *Adv Neural Inf Process Syst*, vol. 32, 2019.

[20] O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky, “Revealing the dark secrets of BERT,” *arXiv preprint arXiv:1908.08593*, 2019.

[21] J. Baan, M. ter Hoeve, M. van der Wees, A. Schuth, and M. de Rijke, “Understanding multi-head attention in abstractive summarization,” *arXiv preprint arXiv:1911.03898*, 2019.

[22] B. Li *et al.*, “Efficient Transformer-based Large Scale Language Representations using Hardware-friendly Block Structured Pruning,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 3187–3199. doi: 10.18653/v1/2020.findings-emnlp.286.

[23] S. Prasanna, A. Rogers, and A. Rumshisky, “When bert plays the lottery, all tickets are winning,” *arXiv preprint arXiv:2005.00561*, 2020.

[24] X. Chen, Y. Cheng, S. Wang, Z. Gan, Z. Wang, and J. Liu, “EarlyBERT: Efficient BERT Training via Early-bird Lottery Tickets,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds., Online: Association for Computational Linguistics, Aug. 2021, pp. 2195–2207. doi: 10.18653/v1/2021.acl-long.171.

[25] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[26] Y. Jin, R. Zhong, S. Long, and J. Zhai, “Efficient Inference for Pruned CNN Models on Mobile Devices With Holistic Sparsity Alignment,” *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–16, 2024, doi: 10.1109/TPDS.2024.3462092.

[27] B. Li *et al.*, “Efficient transformer-based large scale language representations using hardware-friendly block structured pruning,” *arXiv preprint arXiv:2009.08065*, 2020.

[28] A. Fan, E. Grave, and A. Joulin, “Reducing transformer depth on demand with structured dropout,” *arXiv preprint arXiv:1909.11556*, 2019.

[29] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, “Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned,” *arXiv preprint arXiv:1905.09418*, 2019.

[30] J. S. McCarley, R. Chakravarti, and A. Sil, “Structured pruning of a BERT-based question answering model,” *arXiv preprint arXiv:1910.06360*, 2019.

[31] D. Xu, I. E. H. Yen, J. Zhao, and Z. Xiao, “Rethinking Network Pruning–under the Pre-train and Fine-tune Paradigm,” *arXiv preprint arXiv:2104.08682*, 2021.

[32] B. Cui, Y. Li, and Z. Zhang, “Joint structured pruning and dense knowledge distillation for efficient transformer model compression,” *Neurocomputing*, vol. 458, pp. 56–69, Oct. 2021, doi: 10.1016/j.neucom.2021.05.084.

[33] E. Barbierato and A. Gatti, “Toward Green AI: A Methodological Survey of the Scientific Literature,” *IEEE Access*, vol. 12, pp. 23989–24013, 2024, doi: 10.1109/ACCESS.2024.3360705.

[34] M. Zhu and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression,” *arXiv preprint arXiv:1710.01878*, 2017.

[35] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov, “Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, pp. 5797–5808. doi: 10.18653/v1/P19-1580.

[36] D. Molchanov, A. Ashukha, and D. Vetrov, “Variational dropout sparsifies deep neural networks,” in *International Conference on Machine Learning*, 2017, pp. 2498–2507.

[37] A. Jaiswal, S. Liu, T. Chen, and Z. Wang, “The Emergence of Essential Sparsity in Large Pre-trained Models: The Weights that Matter,” Jun. 2023.

[38] Z. Wang, J. Wohlwend, and T. Lei, “Structured Pruning of Large Language Models,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 6151–6162. doi: 10.18653/v1/2020.emnlp-main.496.

[39] M. J. Hasan, F. Rahman, and N. Mohammed, “Graphical Abstract OptimCLM: Optimizing clinical language models for predicting patient outcomes via knowledge distillation, pruning and quantization Admission notes from the MIMIC-III clinical database OptimCLM: Optimizing clinical language models for predicting patient outcomes via knowledge distillation, pruning and quantization OptimCLM: Optimizing clinical language models for predicting patient outcomes via knowledge distillation, pruning and quantization.” [Online]. Available: https://ssrn.com/abstract=4797716

[40] A. Rogers, O. Kovaleva, and A. Rumshisky, “A primer in bertology: What we know about how bert works,” *Trans Assoc Comput Linguist*, vol. 8, pp. 842–866, 2020.

[41] C. Xu and J. McAuley, “A survey on model compression and acceleration for pretrained language models,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, pp. 10566–10575.

[42] N. Smarts, R. Selvaraj, and V. M. Kuthadi, “Ensemble of winning tickets: pruning bidirectional encoder from the transformers attention heads for enhanced model efficiency,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 15, no. 2, p. 2070, Apr. 2025, doi: 10.11591/ijece.v15i2.pp2070-2080.

[43] Y.-A. Lai, G. Lalwani, and Y. Zhang, “Context Analysis for Pre-trained Masked Language Models,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 3789–3804. doi: 10.18653/v1/2020.findings-emnlp.338.

[44] A. Vaswani *et al.*, “Attention is all you need,” *Adv Neural Inf Process Syst*, vol. 30, 2017.

[45] A. Hernández and J. M. Amigó, “Attention Mechanisms and Their Applications to Complex Systems,” *Entropy*, vol. 23, no. 3, p. 283, Feb. 2021, doi: 10.3390/e23030283.

[46] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, “Efficient Content-Based Sparse Attention with Routing Transformers,” *Trans Assoc Comput Linguist*, vol. 9, pp. 53–68, Feb. 2021, doi: 10.1162/tacl_a_00353.

[47] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, “Dynabert: Dynamic bert with adaptive width and depth,” *Adv Neural Inf Process Syst*, vol. 33, pp. 9782–9793, 2020.

[48] J. Campino, “Unleashing the transformers: NLP models detect AI writing in education,” *Journal of Computers in Education*, Jun. 2024, doi: 10.1007/s40692-024-00325-y.

[49] F. A. Gers and E. Schmidhuber, “LSTM recurrent networks learn simple context-free and context-sensitive languages,” *IEEE Trans Neural Netw*, vol. 12, no. 6, pp. 1333–1340, 2001.

[50] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997.

[51] S. Wang and J. Jiang, “Learning natural language inference with LSTM,” *arXiv preprint arXiv:1512.08849*, 2015.

[52] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM networks,” in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2005, pp. 2047–2052.

[53] H. Ghaemi, Z. Alizadehsani, A. Shahraki, and J. M. Corchado, “Transformers in source code generation: A comprehensive survey,” *Journal of Systems Architecture*, vol. 153, p. 103193, Aug. 2024, doi: 10.1016/j.sysarc.2024.103193.

[54] S. Lamsiyah, A. El Mahdaouy, S. E. A. Ouatik, and B. Espinasse, “Unsupervised extractive multi-document summarization method based on transfer learning from BERT multi-task fine-tuning,” *J Inf Sci*, vol. 49, no. 1, pp. 164–182, Feb. 2023, doi: 10.1177/0165551521990616.

[55] P. Su and K. Vijay-Shanker, “Investigation of improving the pre-training and fine-tuning of BERT model for biomedical relation extraction,” *BMC Bioinformatics*, vol. 23, no. 1, p. 120, Dec. 2022, doi: 10.1186/s12859-022-04642-w.

[56] Z. Liu, W. Lin, Y. Shi, and J. Zhao, “A Robustly Optimized BERT Pre-training Approach with Post-training,” 2021, pp. 471–484. doi: 10.1007/978-3-030-84186-7_31.

[57] L. Kryeziu and V. Shehu, “A Survey of Using Unsupervised Learning Techniques in Building Masked Language Models for Low Resource Languages,” in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, IEEE, Jun. 2022, pp. 1–6. doi: 10.1109/MECO55406.2022.9797081.

[58] H. Liu, Y. Perl, and J. Geller, “Concept placement using BERT trained by transforming and summarizing biomedical ontology structure,” *J Biomed Inform*, vol. 112, p. 103607, Dec. 2020, doi: 10.1016/j.jbi.2020.103607.

[59] J. M. Gomez-Perez, R. Denaux, and A. Garcia-Silva, “Understanding Word Embeddings and Language Models,” in *A Practical Guide to Hybrid Natural Language Processing*, Cham: Springer International Publishing, 2020, pp. 17–31. doi: 10.1007/978-3-030-44830-1_3.

[60] H. Wang, J. Li, H. Wu, E. Hovy, and Y. Sun, “Pre-Trained Language Models and Their Applications,” *Engineering*, vol. 25, pp. 51–65, Jun. 2023, doi: 10.1016/j.eng.2022.04.024.

[61] P. Ganesh *et al.*, “Compressing Large-Scale Transformer-Based Models: A Case Study on BERT,” *Trans Assoc Comput Linguist*, vol. 9, pp. 1061–1080, Sep. 2021, doi: 10.1162/tacl_a_00413.

[62] A. Torfi, R. A. Shirvani, Y. Keneshloo, N. Tavaf, and E. A. Fox, “Natural language processing advancements by deep learning: A survey,” *arXiv preprint arXiv:2003.01200*, 2020.

[63] M. A. K. Raiaan *et al.*, “A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges,” *IEEE Access*, vol. 12, pp. 26839–26874, 2024, doi: 10.1109/ACCESS.2024.3365742.

[64] T. Wolf *et al.*, “Transformers: State-of-the-Art Natural Language Processing,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 38–45. doi: 10.18653/v1/2020.emnlp-demos.6.

[65] J. Baan, M. ter Hoeve, M. van der Wees, A. Schuth, and M. de Rijke, “Understanding multi-head attention in abstractive summarization,” *arXiv preprint arXiv:1911.03898*, 2019.

[66] T. Chen *et al.*, “The lottery ticket hypothesis for pre-trained bert networks,” *Adv Neural Inf Process Syst*, vol. 33, pp. 15834–15846, 2020.

[67] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” *Adv Neural Inf Process Syst*, vol. 29, 2016.

[68] J. Li, R. Cotterell, and M. Sachan, “Differentiable subset pruning of transformer heads,” *Trans Assoc Comput Linguist*, vol. 9, pp. 1442–1459, 2021.

[69] Z. Lin, J. Z. Liu, Z. Yang, N. Hua, and D. Roth, “Pruning redundant mappings in transformer models via spectral-normalized identity prior,” *arXiv preprint arXiv:2010.01791*, 2020.

[70] A. Parnami, R. Singh, and T. Joshi, “Pruning Attention Heads of Transformer Models Using A* Search: A Novel Approach to Compress Big NLP Architectures,” *arXiv preprint arXiv:2110.15225*, 2021.

[71] S. Kim *et al.*, “Learned token pruning for transformers,” *arXiv preprint arXiv:2107.00910*, 2021.

[72] H. Wang, Z. Zhang, and S. Han, “Spatten: Efficient sparse attention architecture with cascade token and head pruning,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 97–110.

[73] A. Khetan and Z. Karnin, “schuBERT: Optimizing elements of BERT,” *arXiv preprint arXiv:2005.06628*, 2020.

[74] C. Liu *et al.*, “Progressive neural architecture search,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 19–34.

[75] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, “Efficient neural architecture search via parameters sharing,” in *International conference on machine learning*, 2018, pp. 4095–4104.

[76] I. Singh, H. Zhou, K. Yang, M. Ding, B. Lin, and P. Xie, “Differentially-private federated neural architecture search,” *arXiv preprint arXiv:2006.10559*, 2020.

[77] H. Tsai, J. Ooi, C.-S. Ferng, H. W. Chung, and J. Riesa, “Finding Fast Transformers: One-Shot Neural Architecture Search by Component Composition,” *ArXiv*, vol. abs/2008.06808, 2020, [Online]. Available: https://api.semanticscholar.org/CorpusID:221140129

[78] S. N. Sridhar and A. Sarah, “Undivided Attention: Are Intermediate Layers Necessary for BERT?,” *arXiv preprint arXiv:2012.11881*, 2020.

[79] H. Sajjad, F. Dalvi, N. Durrani, and P. Nakov, “Poor man’s bert: Smaller and faster transformer models,” *arXiv preprint arXiv:2004.03844*, 2020.

[80] C. Liang *et al.*, “Super tickets in pre-trained language models: From model compression to improving generalization,” *arXiv preprint arXiv:2105.12002*, 2021.

[81] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.

[82] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, “Regularization of neural networks using dropconnect,” in *International conference on machine learning*, 2013, pp. 1058–1066.

[83] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, “Deep networks with stochastic depth,” in *European conference on computer vision*, 2016, pp. 646–661.

[84] Y. LeCun, J. Denker, and S. Solla, “Optimal brain damage,” *Adv Neural Inf Process Syst*, vol. 2, 1989.

[85] B. Hassibi and D. Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” *Adv Neural Inf Process Syst*, vol. 5, 1992.

[86] M. A. Gordon, K. Duh, and N. Andrews, “Compressing bert: Studying the effects of weight pruning on transfer learning,” *arXiv preprint arXiv:2002.08307*, 2020.

[87] Y. Mao *et al.*, “Ladabert: Lightweight adaptation of bert through hybrid model compression,” *arXiv preprint arXiv:2004.04124*, 2020.

[88] T. Jiang, D. Wang, F. Zhuang, R. Xie, and F. Xia, “Pruning Pre-trained Language Models Without Fine-Tuning,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2023, pp. 594–605. doi: 10.18653/v1/2023.acl-long.35.

[89] S. Huang *et al.*, “An Automatic and Efficient BERT Pruning for Edge AI Systems,” in *2022 23rd International Symposium on Quality Electronic Design (ISQED)*, IEEE, Apr. 2022, pp. 1–6. doi: 10.1109/ISQED54688.2022.9806197.

[90] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, “Amc: Automl for model compression and acceleration on mobile devices,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 784–800.

[91] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, “Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 4876–4883.

[92] A. Mallya, D. Davis, and S. Lazebnik, “Piggyback: Adapting a single network to multiple tasks by learning to mask weights,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 67–82.

[93] L. L. Zhang *et al.*, “SwiftPruner: Reinforced Evolutionary Pruning for Efficient Ad Relevance,” in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 3654–3663.

[94] F.-M. Guo, S. Liu, F. S. Mungall, X. Lin, and Y. Wang, “Reweighted proximal pruning for large-scale language representation,” *arXiv preprint arXiv:1909.12486*, 2019.

[95] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *arXiv preprint arXiv:1711.05101*, 2017.

[96] E. J. Candes, M. B. Wakin, and S. P. Boyd, “Enhancing sparsity by reweighted ℓ 1 minimization,” *Journal of Fourier analysis and applications*, vol. 14, no. 5, pp. 877–905, 2008.

[97] N. Parikh, S. Boyd, and others, “Proximal algorithms,” *Foundations and trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[98] O. Zafrir, A. Larey, G. Boudoukh, H. Shen, and M. Wasserblat, “Prune once for all: Sparse pre-trained language models,” *arXiv preprint arXiv:2111.05754*, 2021.

[99] E. Kurtic *et al.*, “The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models,” *arXiv preprint arXiv:2203.07259*, 2022.

[100] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush, “Block pruning for faster transformers,” *arXiv preprint arXiv:2109.04838*, 2021.

[101] C. Fan, J. Li, X. Ao, F. Wu, Y. Meng, and X. Sun, “Layer-wise model pruning based on mutual information,” *arXiv preprint arXiv:2108.12594*, 2021.

[102] P. Xia and R. Shin, “Pruning Pretrained Encoders with a Multitask Objective,” *arXiv preprint arXiv:2112.05705*, 2021.

[103] Y. J. Kim and H. H. Awadalla, “Fastformers: Highly efficient transformer models for natural language understanding,” *arXiv preprint arXiv:2010.13382*, 2020.

[104] Z. Yao *et al.*, “LEAP: Learnable Pruning for Transformer-based Models,” *arXiv e-prints*, p. arXiv–2105, 2021.

[105] J. M. Alvarez and M. Salzmann, “Compression-aware training of deep networks,” *Adv Neural Inf Process Syst*, vol. 30, 2017.

[106] Z. Yao, X. Wu, C. Li, S. Youn, and Y. He, “Exploring Post-training Quantization in LLMs from Comprehensive Study to Low Rank Compensation,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, pp. 19377–19385, Mar. 2024, doi: 10.1609/aaai.v38i17.29908.

[107] H. Ryu, S. Lim, and H. Shim, “Memory-Efficient Fine-Tuning for Quantized Diffusion Model,” 2025, pp. 356–372. doi: 10.1007/978-3-031-72640-8_20.

[108] S. Zhang *et al.*, “Opt: Open pre-trained transformer language models,” *arXiv preprint arXiv:2205.01068*, 2022.

[109] X. Wei *et al.*, “Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling,” *arXiv preprint arXiv:2304.09145*, 2023.

[110] H. Bai, L. Hou, L. Shang, X. Jiang, I. King, and M. R. Lyu, “Towards efficient post-training quantization of pre-trained language models,” *Adv Neural Inf Process Syst*, vol. 35, pp. 1405–1418, 2022.

[111] J. Kim *et al.*, “Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization,” *Adv Neural Inf Process Syst*, vol. 36, 2024.

[112] U. Bibi *et al.*, “Advances in Pruning and Quantization for Natural Language Processing,” *IEEE Access*, 2024.

[113] G. Hinton, O. Vinyals, J. Dean, and others, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[114] J. Zhao *et al.*, “Multistage collaborative knowledge distillation from a large language model for semi-supervised sequence generation,” in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 14201–14214.

[115] Z. Liu *et al.*, “Llm-qat: Data-free quantization aware training for large language models,” *arXiv preprint arXiv:2305.17888*, 2023.

[116] Y. Kim and A. M. Rush, “Sequence-level knowledge distillation,” *arXiv preprint arXiv:1606.07947*, 2016.

[117] S. Yao, J. Tan, X. Chen, J. Zhang, X. Zeng, and K. Yang, “ReprBERT: distilling BERT to an efficient representation-based relevance model for e-commerce,” in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4363–4371.

[118] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, “Zeroquant: Efficient and affordable post-training quantization for large-scale transformers,” *Adv Neural Inf Process Syst*, vol. 35, pp. 27168–27183, 2022.

[119] B. Cui, Y. Li, and Z. Zhang, “Joint structured pruning and dense knowledge distillation for efficient transformer model compression,” *Neurocomputing*, vol. 458, pp. 56–69, Oct. 2021, doi: 10.1016/j.neucom.2021.05.084.

[120] Y. Gu, L. Dong, F. Wei, and M. Huang, “MiniLLM: Knowledge distillation of large language models,” in *The Twelfth International Conference on Learning Representations*, 2024.

[121] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, and A. K. Somani, “Neural Architecture Search for Transformers: A Survey,” *IEEE Access*, vol. 10, pp. 108374–108412, 2022, doi: 10.1109/ACCESS.2022.3212767.

[122] L. Pouy, F. Khenfri, P. Leserf, C. Mraidha, and C. Larouci, “An Approach for Efficient Neural Architecture Search Space Definition,” *arXiv preprint arXiv:2310.17669*, 2023.

[123] J. Xu *et al.*, “NAS-BERT: Task-agnostic and adaptive-size BERT compression with neural architecture search,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1933–1943.

[124] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur, “Reinforcement learning for neural architecture search: A review,” *Image Vis Comput*, vol. 89, pp. 57–66, 2019.

[125] P. Ren *et al.*, “A comprehensive survey of neural architecture search: Challenges and solutions,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[126] A. Klein, J. Golebiowski, X. Ma, V. Perrone, and C. Archambeau, “Structural pruning of large language models via neural architecture search,” 2023.

[127] T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann, “Evolutionary Algorithms,” *WIREs Data Mining and Knowledge Discovery*, vol. 4, no. 3, pp. 178–195, May 2014, doi: 10.1002/widm.1124.

[128] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search,” *arXiv preprint arXiv:1806.09055*, 2018.

[129] S. Liu, H. Zhang, and Y. Jin, “A survey on computationally efficient neural architecture search,” *Journal of Automation and Intelligence*, vol. 1, no. 1, p. 100002, Dec. 2022, doi: 10.1016/j.jai.2022.100002.

[130] N. Klyuchnikov *et al.*, “Nas-bench-nlp: neural architecture search benchmark for natural language processing,” *IEEE Access*, vol. 10, pp. 45736–45747, 2022.

[131] D. Chen *et al.*, “Adabert: Task-adaptive bert compression with differentiable neural architecture search,” *arXiv preprint arXiv:2001.04246*, 2020.

[132] Y. Yang, J. Zhan, W. Jiang, Y. Jiang, and A. Yu, “Neural architecture search for resource constrained hardware devices: A survey,” *IET Cyber-Physical Systems: Theory & Applications*, vol. 8, no. 3, pp. 149–159, Sep. 2023, doi: 10.1049/cps2.12058.

[133] S. Pletenev, D. Moskovskiy, V. Chekalina, M. Seleznyov, S. Zagoruyko, and A. Panchenko, “Transformers Compression: A Study of Matrix Decomposition Methods Using Fisher Information,” 2024, pp. 36–48. doi: 10.1007/978-3-031-54534-4_3.

[134] B. Zhang, D. Cheng, Y. Zhang, F. Liu, and J. Tian, “Lossless Model Compression via Joint Low-Rank Factorization Optimization,” *arXiv preprint arXiv:2412.06867*, 2024.

[135] S. Ren and K. Q. Zhu, “Low-rank prune-and-factorize for language model compression,” *arXiv preprint arXiv:2306.14152*, 2023.

[136] S. Pletenev, V. Chekalina, D. Moskovskiy, M. Seleznev, S. Zagoruyko, and A. Panchenko, “A Computational Study of Matrix Decomposition Methods for Compression of Pre-trained

Transformers," in *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*, 2023, pp. 723–742.

[137] H. Hajimolahoseini, M. Rezagholizadeh, V. Partovinia, M. Tahaei, O. M. Awad, and Y. Liu, "Compressing pre-trained language models using progressive low rank decomposition," *Adv Neural Inf Process Syst*, 2021.

[138] Y. Li *et al.*, "Losparse: Structured compression of large language models based on low-rank and sparse approximation," in *International Conference on Machine Learning*, 2023, pp. 20336–20350.

[139] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[140] J. Hu, *Applying Deep Compression to Enable Spoken Language Edge Inference*. McGill University (Canada), 2022.

[141] X. Jiao *et al.*, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[142] S. Liu *et al.*, "Foldgpt: Simple and effective large language model compression scheme," *arXiv preprint arXiv:2407.00928*, 2024.

[143] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[144] P. V. Dantas, W. Sabino da Silva, L. C. Cordeiro, and C. B. Carvalho, "A comprehensive review of model compression techniques in machine learning," *Applied Intelligence*, vol. 54, no. 22, pp. 11804–11844, Nov. 2024, doi: 10.1007/s10489-024-05747-w.

[145] P. Liu, Z.-F. Gao, W. X. Zhao, Z.-Y. Xie, Z.-Y. Lu, and J.-R. Wen, "Enabling lightweight fine-tuning for pre-trained language model compression based on matrix product operators," *arXiv preprint arXiv:2106.02205*, 2021.

[146] P. Jin, B. Zhu, L. Yuan, and S. Yan, "MoH: Multi-Head Attention as Mixture-of-Head Attention," *arXiv preprint arXiv:2410.11842*, 2024.

[147] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "DeeBERT: Dynamic early exiting for accelerating BERT inference," *arXiv preprint arXiv:2004.12993*, 2020.

[148] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, "Bert loses patience: Fast and robust inference with early exit," *Adv Neural Inf Process Syst*, vol. 33, pp. 18330–18341, 2020.

[149] J. He *et al.*, "DE ^ 3-BERT: Distance-Enhanced Early Exiting for BERT based on Prototypical Networks," *arXiv preprint arXiv:2402.05948*, 2024.

[150] K. Liao, Y. Zhang, X. Ren, Q. Su, X. Sun, and B. He, “A global past-future early exit method for accelerating inference of pre-trained language models,” in *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2021, pp. 2013–2023.

[151] H. Rahmath P, V. Srivastava, K. Chaurasia, R. G. Pacheco, and R. S. Couto, “Early-Exit Deep Neural Network-A Comprehensive Survey,” *ACM Comput Surv*, vol. 57, no. 3, pp. 1–37, 2024.

[152] N. Passalis and A. Tefas, “Training lightweight deep convolutional neural networks using bag-of-features pooling,” *IEEE Trans Neural Netw Learn Syst*, vol. 30, no. 6, pp. 1705–1715, 2018.

[153] J. Xiao, Z. Li, L. Yang, Y. Mei, and Q. Gu, “TTAQ: Towards Stable Post-training Quantization in Continuous Domain Adaptation,” *arXiv preprint arXiv:2412.09899*, 2024.

[154] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, “Smoothquant: Accurate and efficient post-training quantization for large language models,” in *International Conference on Machine Learning*, 2023, pp. 38087–38099.

[155] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, “I-bert: Integer-only bert quantization,” in *International conference on machine learning*, 2021, pp. 5506–5518.

[156] Y. Zheng, Y. Chen, B. Qian, X. Shi, Y. Shu, and J. Chen, “A Review on Edge Large Language Models: Design, Execution, and Applications,” *arXiv preprint arXiv:2410.11845*, 2024.

[157] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” *arXiv preprint arXiv:1910.01108*, 2019.

[158] S. Sun, Y. Cheng, Z. Gan, and J. Liu, “Patient knowledge distillation for bert model compression,” *arXiv preprint arXiv:1908.09355*, 2019.

[159] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, “Mobilebert: a compact task-agnostic bert for resource-limited devices,” *arXiv preprint arXiv:2004.02984*, 2020.

[160] Z. Ding, G. Jiang, S. Zhang, L. Guo, and W. Lin, “SKDBERT: Compressing BERT via stochastic knowledge distillation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, pp. 7414–7422.

[161] B. G.M. and G. N. Pillai, “Sequential node search for faster neural architecture search,” *Knowl Based Syst*, vol. 300, p. 112145, Sep. 2024, doi: 10.1016/j.knosys.2024.112145.

[162] A. Paraskeva, J. P. Reis, S. Verberne, and J. N. van Rijn, “Resource-constrained Neural Architecture Search on Language Models: A Case Study,” in *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ ICML 2024)*,

[163] M. Ben Noach and Y. Goldberg, “Compressing pre-trained language models by matrix decomposition,” in *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, 2020, pp. 884–889.

[164] Y. Xu, C. Song, Y. Sun, and S. Yu, “Advances and Challenges in Large Model Compression: A Survey,” in *Proceedings of the 2024 Guangdong-Hong Kong-Macao Greater Bay Area International Conference on Digital Economy and Artificial Intelligence*, 2024, pp. 421–426.

[165] Y.-C. Hsu, T. Hua, S. Chang, Q. Lou, Y. Shen, and H. Jin, “Language model compression with weighted low-rank factorization,” *arXiv preprint arXiv:2207.00112*, 2022.

[166] P. Chen, H.-F. Yu, I. Dhillon, and C.-J. Hsieh, “Drone: Data-aware low-rank compression for large nlp models,” *Adv Neural Inf Process Syst*, vol. 34, pp. 29321–29334, 2021.

[167] M. Cho *et al.*, “eDKM: An Efficient and Accurate Train-time Weight Clustering for Large Language Models,” *IEEE Computer Architecture Letters*, 2024.

[168] J. C. Hu, R. Cavicchioli, G. Berardinelli, and A. Capotondi, “ShareBERT: Embeddings Are Capable of Learning Hidden Layers,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024, pp. 18225–18233.

[169] X. Ma *et al.*, “A tensorized transformer for language modeling,” *Adv Neural Inf Process Syst*, vol. 32, 2019.

[170] M. Xu, Y. L. Xu, and D. P. Mandic, “TensorGPT: Efficient Compression of Large Language Models Based on Tensor-Train Decomposition,” *arXiv. org*, 2023.

[171] J. Xin, R. Tang, Y. Yu, and J. Lin, “BERxiT: Early exiting for BERT with better fine-tuning and extension to regression,” in *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume*, 2021, pp. 91–104.

[172] X. Li, Y. Shao, T. Sun, H. Yan, X. Qiu, and X. Huang, “Accelerating bert inference for sequence labeling via early-exit,” *arXiv preprint arXiv:2105.13878*, 2021.

[173] J. Frankle and M. Carbin, “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks,” *arXiv: Learning*, 2018, [Online]. Available: https://api.semanticscholar.org/CorpusID:53388625

[174] T. Chen *et al.*, “The lottery ticket hypothesis for pre-trained bert networks,” *Adv Neural Inf Process Syst*, vol. 33, pp. 15834–15846, 2020.

[175] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, “Stabilizing the lottery ticket hypothesis,” *arXiv preprint arXiv:1903.01611*, 2019.

[176] R. Cheong and R. Daniel, “transformers. zip: Compressing Transformers with Pruning and Quantization,” *Technical report, tech. rep., Stanford University, Stanford, California*, 2019.

[177] R. Moura, J. Carvalho, A. Plastino, and A. Paes, “Less is more: Pruning BERTweet architecture in Twitter sentiment analysis,” *Inf Process Manag*, vol. 61, no. 4, p. 103688, Jul. 2024, doi: 10.1016/j.ipm.2024.103688.

[178] D. Qiu and B. Yang, “Text summarization based on multi-head self-attention mechanism and pointer network,” *Complex & Intelligent Systems*, vol. 8, no. 1, pp. 555–567, Feb. 2022, doi: 10.1007/s40747-021-00527-2.

[179] P. Rajapaksha and N. Crespi, “Explainable Attention Pruning: A Metalearning-Based Approach,” *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 6, pp. 2505–2516, Jun. 2024, doi: 10.1109/TAI.2024.3363686.

[180] Z. Zhang, F. Qi, Z. Liu, Q. Liu, and M. Sun, “Know what you don’t need: Single-Shot Meta-Pruning for attention heads,” *AI Open*, vol. 2, pp. 36–42, 2021, doi: 10.1016/j.aiopen.2021.05.003.

[181] I. Rugina, R. Dangovski, L. Jing, P. Nakov, and M. Soljačić, “Data-Informed Global Sparseness in Attention Mechanisms for Deep Neural Networks,” *arXiv preprint arXiv:2012.02030*, 2020.

[182] M. Behnke and K. Heafield, “Losing Heads in the Lottery: Pruning Transformer Attention in Neural Machine Translation,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 2664–2674. doi: 10.18653/v1/2020.emnlp-main.211.

[183] A. Mohammed and R. Kora, “A comprehensive review on ensemble deep learning: Opportunities and challenges,” *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 2, pp. 757–774, Feb. 2023, doi: 10.1016/j.jksuci.2023.01.014.

[184] I. D. Mienye and Y. Sun, “A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects,” *IEEE Access*, vol. 10, pp. 99129–99149, 2022, doi: 10.1109/ACCESS.2022.3207287.

[185] J. Risch and R. Krestel, “Bagging BERT models for robust aggression identification,” in *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*, 2020, pp. 55–61.

[186] L. Breiman, “Bagging predictors,” *Mach Learn*, vol. 24, no. 2, pp. 123–140, Aug. 1996, doi: 10.1007/BF00058655.

[187] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, “Ensemble deep learning: A review,” *Eng Appl Artif Intell*, vol. 115, p. 105151, 2022.

[188] H. Katpally and A. Bansal, “Ensemble Learning on Deep Neural Networks for Image Caption Generation,” in *2020 IEEE 14th International Conference on Semantic Computing (ICSC)*, IEEE, Feb. 2020, pp. 61–68. doi: 10.1109/ICSC.2020.00016.

[189] Y. Guo, J. Feng, B. Jiao, N. Cui, S. Yang, and Z. Yu, “A dual evolutionary bagging for class imbalance learning,” *Expert Syst Appl*, vol. 206, p. 117843, Nov. 2022, doi: 10.1016/j.eswa.2022.117843.

[190] G. Cassales, H. Gomes, A. Bifet, B. Pfahringer, and H. Senger, “Improving the performance of bagging ensembles for data streams through mini-batching,” *Inf Sci (N Y)*, vol. 580, pp. 260–282, Nov. 2021, doi: 10.1016/j.ins.2021.08.085.

[191] N. Altman and M. Krzywinski, “Ensemble methods: bagging and random forests,” *Nat Methods*, vol. 14, no. 10, pp. 933–934, Oct. 2017, doi: 10.1038/nmeth.4438.

[192] P. Yang, Y. Hwa Yang, B. B. Zhou, and A. Y. Zomaya, “A Review of Ensemble Methods in Bioinformatics,” *Curr Bioinform*, vol. 5, no. 4, pp. 296–308, Dec. 2010, doi: 10.2174/157489310794072508.

[193] A. Chikhi, S. S. Mohammadi Ziabari, and J.-W. van Essen, “A Comparative Study of Traditional, Ensemble and Neural Network-Based Natural Language Processing Algorithms,” *Journal of Risk and Financial Management*, vol. 16, no. 7, p. 327, Jul. 2023, doi: 10.3390/jrfm16070327.

[194] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[195] A. Al Ali, A. M. Khedr, M. El-Bannany, and S. Kanakkayil, “A Powerful Predicting Model for Financial Statement Fraud Based on Optimized XGBoost Ensemble Learning Technique,” *Applied Sciences*, vol. 13, no. 4, p. 2272, Feb. 2023, doi: 10.3390/app13042272.

[196] X. Zhang, S. Liu, X. Wang, and Y. Li, “A fragmented neural network ensemble method and its application to image classification,” *Sci Rep*, vol. 14, no. 1, p. 2291, Jan. 2024, doi: 10.1038/s41598-024-52945-0.

[197] H. Drucker, C. Cortes, L. D. Jackel, Y. LeCun, and V. Vapnik, “Boosting and Other Ensemble Methods,” *Neural Comput*, vol. 6, no. 6, pp. 1289–1301, Nov. 1994, doi: 10.1162/neco.1994.6.6.1289.

[198] L. Jovanovic, “Natural Language Processing and AdaBoost Optimized by Modified,” *Innovations and Advances in Cognitive Systems: ICIACS 2024, Volume 2*, vol. 2, p. 446.

[199] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz, “A comparative analysis of gradient boosting algorithms,” *Artif Intell Rev*, vol. 54, no. 3, pp. 1937–1967, Mar. 2021, doi: 10.1007/s10462-020-09896-5.

[200] A. Odeh, Q. A. Al-Haija, A. Aref, and A. A. Taleb, “Comparative Study of CatBoost, XGBoost, and LightGBM for Enhanced URL Phishing Detection: A Performance Assessment,” *Journal of*

Internet Services and Information Security, vol. 13, no. 4, pp. 1–11, Dec. 2023, doi: 10.58346/JISIS.2023.I4.001.

[201] M. A. Muslim *et al.*, "New model combination meta-learner to improve accuracy prediction P2P lending with stacking ensemble learning," *Intelligent Systems with Applications*, vol. 18, p. 200204, May 2023, doi: 10.1016/j.iswa.2023.200204.

[202] A. Ghosh, D. Dutta, and T. Moitra, "A Neural Network Framework to Generate Caption from Images," 2020, pp. 171–180. doi: 10.1007/978-981-13-7403-6_17.

[203] W. van Loon, M. Fokkema, B. Szabo, and M. de Rooij, "View selection in multi-view stacking: choosing the meta-learner," *Adv Data Anal Classif*, Apr. 2024, doi: 10.1007/s11634-024-00587-5.

[204] C. Ju, A. Bibaut, and M. van der Laan, "The relative performance of ensemble methods with deep convolutional neural networks for image classification," *J Appl Stat*, vol. 45, no. 15, pp. 2800–2818, Nov. 2018, doi: 10.1080/02664763.2018.1441383.

[205] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, Jan. 1992, doi: 10.1016/S0893-6080(05)80023-1.

[206] A. I. Naimi and L. B. Balzer, "Stacked generalization: an introduction to super learning," *Eur J Epidemiol*, vol. 33, no. 5, pp. 459–464, May 2018, doi: 10.1007/s10654-018-0390-z.

[207] M. J. van der Laan, E. C. Polley, and A. E. Hubbard, "Super Learner," *Stat Appl Genet Mol Biol*, vol. 6, no. 1, Jan. 2007, doi: 10.2202/1544-6115.1309.

[208] J. Jia, W. Liang, and Y. Liang, "A review of hybrid and ensemble in deep learning for natural language processing," *arXiv preprint arXiv:2312.05589*, 2023.

[209] Y. Bian, J. Huang, X. Cai, J. Yuan, and K. Church, "On attention redundancy: A comprehensive study," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 930–945.

[210] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter, "A simple and effective pruning approach for large language models," *arXiv preprint arXiv:2306.11695*, 2023.

[211] W. Held and D. Yang, "Shapley Head Pruning: Identifying and Removing Interference in Multilingual Transformers," *arXiv preprint arXiv:2210.05709*, 2022.

[212] C. Ju, A. Bibaut, and M. van der Laan, "The relative performance of ensemble methods with deep convolutional neural networks for image classification," *J Appl Stat*, vol. 45, no. 15, pp. 2800–2818, Nov. 2018, doi: 10.1080/02664763.2018.1441383.

[213] Y. I. Alzoubi and A. Mishra, "Green artificial intelligence initiatives: Potentials and challenges," *J Clean Prod*, vol. 468, p. 143090, Aug. 2024, doi: 10.1016/j.jclepro.2024.143090.

[214] V. Bolón-Canedo, L. Morán-Fernández, B. Cancela, and A. Alonso-Betanzos, “A review of green artificial intelligence: Towards a more sustainable future,” *Neurocomputing*, vol. 599, p. 128096, Sep. 2024, doi: 10.1016/j.neucom.2024.128096.

[215] A. Bin Rashid and M. A. K. Kausik, “AI revolutionizing industries worldwide: A comprehensive overview of its diverse applications,” *Hybrid Advances*, vol. 7, p. 100277, Dec. 2024, doi: 10.1016/j.hybadv.2024.100277.

[216] P. Rivas *et al.*, “AI ethics for earth sciences,” in *Artificial Intelligence in Earth Science*, Elsevier, 2023, pp. 379–396. doi: 10.1016/B978-0-323-91737-7.00007-4.

[217] L. Waltersmann, S. Kiemel, J. Stuhlsatz, A. Sauer, and R. Miehe, “Artificial Intelligence Applications for Increasing Resource Efficiency in Manufacturing Companies—A Comprehensive Review,” *Sustainability*, vol. 13, no. 12, p. 6689, Jun. 2021, doi: 10.3390/su13126689.

[218] A. Shahid and M. Mushtaq, “A Survey Comparing Specialized Hardware And Evolution In TPUs For Neural Networks,” in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, IEEE, Nov. 2020, pp. 1–6. doi: 10.1109/INMIC50486.2020.9318136.

[219] A. K. Jakkani, “Enhancing Urban Sustainability through AI-Driven Energy Efficiency Strategies in Cloud-Enabled Smart Cities,” *J. Energy Eng. Thermodyn*, vol. 4, pp. 1–13, 2024.

[220] A. Tabbakh, L. Al Amin, M. Islam, G. M. I. Mahmud, I. K. Chowdhury, and M. S. H. Mukta, “Towards sustainable AI: a comprehensive framework for Green AI,” *Discover Sustainability*, vol. 5, no. 1, p. 408, Nov. 2024, doi: 10.1007/s43621-024-00641-4.

[221] D. B. Olawade, O. Z. Wada, A. C. David-Olawade, O. Fapohunda, A. O. Ige, and J. Ling, “Artificial intelligence potential for net zero sustainability: Current evidence and prospects,” *Next Sustainability*, vol. 4, p. 100041, 2024, doi: 10.1016/j.nxsust.2024.100041.

[222] E. Kurtic, T. Hoefler, and D. Alistarh, “How to prune your language model: Recovering accuracy on the" sparsity may cry” benchmark, 2023,” *URL https://arxiv. org/abs/2312.13547*.

[223] M. Gupta *et al.*, “Is complexity required for neural network pruning? a case study on global magnitude pruning,” in *2024 IEEE Conference on Artificial Intelligence (CAI)*, 2024, pp. 747–754.

[224] Z. Li *et al.*, “Train big, then compress: Rethinking model size for efficient training and inference of transformers,” in *International Conference on machine learning*, 2020, pp. 5958–5968.

[225] I. M. Ali, “A guide for positivist research paradigm: From philosophy to methodology,” *Idealogy Journal*, vol. 9, no. 2, 2024.

[226] M. T. Holden and P. Lynch, “Choosing the appropriate methodology: Understanding research philosophy,” *The marketing review*, vol. 4, no. 4, pp. 397–409, 2004.

[227] G. Menghani, “Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better,” *ACM Comput Surv*, vol. 55, no. 12, pp. 1–37, Dec. 2023, doi: 10.1145/3578938.

[228] T. F. Guerin, “Sustainable Technology Development BY PAUL WEAVER, LEO JANSEN, GEERT VAN SPIEGEL, EGBERT VAN SPIEGEL AND PHILIP VERGRAGT 256 pp., 24.0 16.0 2.0 cm, ISBN 1 874719 09 8 hardback, GB£ 35.00, Sheffield, UK: Greenleaf Publishing, 2000,” *Environ Conserv*, vol. 28, no. 4, pp. 378–387, 2001.

[229] B. Kitchenham and P. Brereton, “A systematic review of systematic review process research in software engineering,” *Inf Softw Technol*, vol. 55, no. 12, pp. 2049–2075, Dec. 2013, doi: 10.1016/j.infsof.2013.07.010.

[230] S. Sivarajkumar *et al.*, “Clinical Information Retrieval: A Literature Review,” *J Healthc Inform Res*, vol. 8, no. 2, pp. 313–352, Jun. 2024, doi: 10.1007/s41666-024-00159-4.

[231] J. Shao and J. Zhang, “Communication-Computation Trade-off in Resource-Constrained Edge Inference,” *IEEE Communications Magazine*, vol. 58, no. 12, pp. 20–26, Dec. 2020, doi: 10.1109/MCOM.001.2000373.

[232] R. Singh and S. S. Gill, “Edge AI: A survey,” *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023, doi: 10.1016/j.iotcps.2023.02.004.

[233] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, “Efficient Transformers: A Survey,” *ACM Comput Surv*, vol. 55, no. 6, pp. 1–28, Jun. 2023, doi: 10.1145/3530811.

[234] M. Cho, A. Joshi, and C. Hegde, “ESPN: Extremely Sparse Pruned Networks,” in *2021 IEEE Data Science and Learning Workshop (DSLW)*, IEEE, Jun. 2021, pp. 1–8. doi: 10.1109/DSLW51110.2021.9523404.

[235] Q. Zhang *et al.*, “Platon: Pruning large transformer models with upper confidence bound of weight importance,” in *International Conference on Machine Learning*, 2022, pp. 26809–26823.

[236] R. Zheng *et al.*, “Robust Lottery Tickets for Pre-trained Language Models,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 2211–2224. doi: 10.18653/v1/2022.acl-long.157.

[237] S. Kobayashi, S. Kiyono, J. Suzuki, and K. Inui, “Diverse Lottery Tickets Boost Ensemble from a Single Pretrained Model,” in *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, A. Fan, S. Ilic, T. Wolf, and M. Gallé, Eds., virtual+Dublin: Association for Computational Linguistics, May 2022, pp. 42–50. doi: 10.18653/v1/2022.bigscience-1.4.

[238] Z. Gong *et al.*, “Finding the Dominant Winning Ticket in Pre-Trained Language Models,” in *Findings of the Association for Computational Linguistics: ACL 2022*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2022, pp. 1459–1472. doi: 10.18653/v1/2022.findings-acl.115.

[239] A. Bitar, R. Rosales, and M. Paulitsch, “Gradient-based feature-attribution explainability methods for spiking neural networks,” *Front Neurosci*, vol. 17, Sep. 2023, doi: 10.3389/fnins.2023.1153999.

[240] C. Yeh, Y. Chen, A. Wu, C. Chen, F. Viégas, and M. Wattenberg, “AttentionViz: A Global View of Transformer Attention,” *IEEE Trans Vis Comput Graph*, pp. 1–11, 2023, doi: 10.1109/TVCG.2023.3327163.

[241] J. Liang and R. Liu, “Stacked denoising autoencoder and dropout together to prevent overfitting in deep neural network,” in *2015 8th International Congress on Image and Signal Processing (CISP)*, IEEE, Oct. 2015, pp. 697–701. doi: 10.1109/CISP.2015.7407967.

[242] A. Li, M. Markovic, P. Edwards, and G. Leontidis, “Model pruning enables localized and efficient federated learning for yield forecasting and data sharing,” *Expert Syst Appl*, vol. 242, p. 122847, May 2024, doi: 10.1016/j.eswa.2023.122847.

[243] L. Lu and B. Lyu, “Reducing energy consumption of Neural Architecture Search: An inference latency prediction framework,” *Sustain Cities Soc*, vol. 67, p. 102747, Apr. 2021, doi: 10.1016/j.scs.2021.102747.

[244] H. Cai *et al.*, “Enable Deep Learning on Mobile Devices: Methods, Systems, and Applications,” *ACM Transact Des Autom Electron Syst*, vol. 27, no. 3, pp. 1–50, May 2022, doi: 10.1145/3486618.

[245] S.-K. Yeom *et al.*, “Pruning by explaining: A novel criterion for deep neural network pruning,” *Pattern Recognit*, vol. 115, p. 107899, Jul. 2021, doi: 10.1016/j.patcog.2021.107899.

[246] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, “Green AI,” *Commun ACM*, vol. 63, no. 12, pp. 54–63, Nov. 2020, doi: 10.1145/3381831.

[247] J. Xiao, P. Li, J. Nie, and Z. Tang, “SEVEN: Pruning Transformer Model by Reserving Sentinels,” in *2024 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Jun. 2024, pp. 1–8. doi: 10.1109/IJCNN60899.2024.10651013.

[248] H. Zhang, X. XiaolongShi, J. Sun, and G. Sun, “Structured Pruning for Large Language Models Using Coupled Components Elimination and Minor Fine-tuning,” in *Findings of the Association for Computational Linguistics: NAACL 2024*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2024, pp. 1–12. doi: 10.18653/v1/2024.findings-naacl.1.

[249] Y. Mao *et al.*, “Ladabert: Lightweight adaptation of bert through hybrid model compression,” *arXiv preprint arXiv:2004.04124*, 2020.

[250] M. S. Sayeed, V. Mohan, and K. S. Muthu, “BERT: A Review of Applications in Sentiment Analysis,” *HighTech and Innovation Journal*, vol. 4, no. 2, pp. 453–462, Jun. 2023, doi: 10.28991/HIJ-2023-04-02-015.

Appendices

Appendix 1: Efficient Model Pruning Optimization Algorithm Code

```
from transformers import Trainer, TrainingArguments, DistilBertForSequenceClassification, DistilBertTokenizer
from datasets import load_dataset
import numpy as np
import torch
import time
import random
from torch.utils.data import DataLoader

# Load SST-2 dataset
dataset = load_dataset("glue", "sst2")

# Initialize the DistilBERT model and tokenizer
model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")
tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")

# Tokenization function
def tokenize_function(examples):
    return tokenizer(examples["sentence"], padding="max_length", truncation=True)

# Tokenizing the datasets
tokenized_datasets = dataset.map(tokenize_function, batched=True)

# Compute Metrics function (example)
def compute_metrics(p):
    logits, labels = p
    predictions = np.argmax(logits, axis=-1)
    accuracy = (predictions == labels).mean()
    return {"accuracy": accuracy}

# Placeholder functions for computational cost, inference time, and energy consumption
```

```
def compute_computational_cost(model):
    # Placeholder: Let's assume the cost is proportional to the number of parameters
    return sum(p.numel() for p in model.parameters())  # Count the total number of
parameters

def compute_inference_time(model, input_data):
    # Placeholder: Simulate inference time by random values
    start_time = time.time()
    _ = model(input_data)
    return time.time() - start_time

def compute_energy_consumption(model):
    # Placeholder: Estimate energy consumption based on model size
    return sum(p.numel() for p in model.parameters()) * random.uniform(0.01, 0.05)  #
Random factor for energy

def compute_accuracy(model, dataset):
    # Compute accuracy on the SST-2 dataset
    model.eval()
    correct = 0
    total = 0
    for batch in DataLoader(dataset, batch_size=8, shuffle=False):
        inputs = batch["input_ids"].to(model.device)
        labels = batch["label"].to(model.device)
        with torch.no_grad():
            outputs = model(inputs)
        _, predicted = torch.max(outputs.logits, dim=-1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)
    return correct / total

def prune_model_weights(model, pruning_ratio):
    """
    Prune the model's weights by removing the smallest weights according to the pruning
ratio.
    """
    pruned_model = model
    for name, param in pruned_model.named_parameters():
        if "weight" in name and param.requires_grad:
            flattened_weights = param.abs().flatten()
            threshold_index = int(len(flattened_weights) * pruning_ratio)
            sorted_weights, _ = torch.sort(flattened_weights)
            threshold = sorted_weights[threshold_index]
```

```
            param.data[param.abs() < threshold] = 0
    return pruned_model

def optimize_pruning(model, target_pruning_ratio=0.7, epsilon=0.05):
    # Step 1: Initialize original model weights
    A_W = compute_accuracy(model, tokenized_datasets["validation"])  # Accuracy of the
original model
    C_W = compute_computational_cost(model)  # Computational cost of the original model
    T_W = compute_inference_time(model, tokenized_datasets["validation"])  # Inference
time of the original model
    E_W = compute_energy_consumption(model)  # Energy consumption of the original
model
    print(f"Original Accuracy: {A_W}, Cost: {C_W}, Time: {T_W}, Energy: {E_W}")

    # Step 2: Prune the model
    pruned_model = prune_model_weights(model, target_pruning_ratio)
    A_W_prime = compute_accuracy(pruned_model, tokenized_datasets["validation"])  #
Accuracy of the pruned model
    C_W_prime = compute_computational_cost(pruned_model)  # Computational cost of the
pruned model
    T_W_prime = compute_inference_time(pruned_model,
tokenized_datasets["validation"])  # Inference time of the pruned model
    E_W_prime = compute_energy_consumption(pruned_model)  # Energy consumption of
the pruned model
    print(f"Pruned Accuracy: {A_W_prime}, Cost: {C_W_prime}, Time: {T_W_prime}, Energy:
{E_W_prime}")

    # Step 3: Verify accuracy
    if A_W_prime >= A_W - epsilon:
        print(f"Accuracy after pruning is within acceptable range (>= {A_W - epsilon})")
    else:
        print("Accuracy after pruning is too low, stopping optimization.")
        return None  # Return None to indicate pruning was unsuccessful

    # Step 4: Measure efficiency
    if C_W_prime < C_W and T_W_prime < T_W and E_W_prime < E_W:
        print("Model efficiency improved!")
    else:
        print("Model efficiency did not improve.")

    # Step 5: Optimize pruning ratio if necessary (optional)
    best_pruned_model = pruned_model
    best_accuracy = A_W_prime
```

```
    best_pruning_ratio = target_pruning_ratio

    for i in range(5):  # Run for a few iterations to optimize pruning ratio
        target_pruning_ratio += 0.05  # Increment pruning ratio
        pruned_model = prune_model_weights(model, target_pruning_ratio)
        A_W_prime = compute_accuracy(pruned_model, tokenized_datasets["validation"])

        if A_W_prime >= A_W - epsilon and A_W_prime > best_accuracy:
            best_accuracy = A_W_prime
            best_pruned_model = pruned_model
            best_pruning_ratio = target_pruning_ratio

    print(f"Best Pruning Ratio: {best_pruning_ratio}, Best Accuracy: {best_accuracy}")

    return best_pruned_model, best_accuracy, C_W_prime, T_W_prime, E_W_prime


# Define training arguments
training_args = TrainingArguments(
    output_dir="custom_sst2_distilbert",  # Directory to save the model
    load_best_model_at_end=True,
    num_train_epochs=3,
    eval_strategy="steps",  # Updated to eval_strategy
    eval_steps=100,
    save_steps=100,
    metric_for_best_model="accuracy",
    save_total_limit=10,
)

# Create trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
)

# Train the model
trainer.train()

# After training, optimize pruning
```

```
best_pruned_model, best_accuracy, optimized_cost, optimized_time, optimized_energy =
optimize_pruning(
    model,
    target_pruning_ratio=0.7,  # Set 70% pruning
    epsilon=0.05  # Acceptable accuracy loss threshold
)

if best_pruned_model:
    print("Optimization successful!")
    # Save the pruned model
    best_pruned_model.save_pretrained("best_pruned_model")
else:
    print("Optimization failed!")
```

Appendix 2: pruning BERT attention heads_accuracy metrics.

```
import torch
import torch.optim as optim
import torch.nn as nn
import matplotlib.pyplot as plt
from transformers import (
    DistilBertForSequenceClassification,
    Trainer,
    TrainingArguments,
    DistilBertTokenizer,
    DataCollatorWithPadding,
)
from sklearn.metrics import accuracy_score
import wandb
import os
from datasets import DatasetDict, load_dataset

# Initialize wandb
wandb.init(settings=wandb.Settings(start_method="fork"))

# Define the compute_metrics function
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = logits.argmax(-1)
    accuracy = accuracy_score(labels, predictions)
```

```
    return {"eval_accuracy": accuracy}

# Define the ensemble_bert function
def ensemble_bert(num_iterations, learning_rate, batch_size, convergence_threshold):
    ensemble_set = []
    for iteration in range(num_iterations):
        # Initialize the model
        ticket = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")

        # Define and initialize training arguments
        training_args = TrainingArguments(
            output_dir="./results",
            num_train_epochs=3,
            per_device_train_batch_size=batch_size,
            per_device_eval_batch_size=batch_size,
            learning_rate=learning_rate,
            weight_decay=0.01,
            logging_dir='./logs',
            evaluation_strategy="epoch",
            save_strategy="epoch",
            load_best_model_at_end=True,
            metric_for_best_model="eval_accuracy",
        )

        # Initialize the Trainer
        trainer = Trainer(
            model=ticket,
            args=training_args,
            train_dataset=tokenized_datasets["train"],
            eval_dataset=tokenized_datasets["validation"],
            data_collator=data_collator,
            tokenizer=tokenizer,
            compute_metrics=compute_metrics,
        )

        # Ensure the dataset supports __len__ or set max_steps
        if not hasattr(tokenized_datasets["train"], "__len__"):
            training_args.max_steps = 1000  # Set this value as per your dataset size

        trainer.train()

        # Evaluate and check for convergence
        predictions = trainer.predict(tokenized_datasets["test"])
```

```
        accuracy = accuracy_score(predictions.label_ids, predictions.predictions.argmax(-1))
        if accuracy > convergence_threshold:
            ensemble_set.append(ticket)

            # Log metrics to wandb
            wandb.log({"iteration": iteration, "accuracy": accuracy})

    return ensemble_set

if __name__ == "__main__":
    # Set up wandb configuration
    wandb_config = {
        "num_iterations": 5,
        "learning_rate": 2e-5,
        "batch_size": 16,
        "convergence_threshold": 0.001,
    }
    wandb.config.update(wandb_config)

    # Load the dataset
    raw_datasets = load_dataset("glue", "mrpc")
    tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
    data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

    # Preprocess the dataset
    def preprocess_function(examples):
        return tokenizer(examples["sentence1"], examples["sentence2"], truncation=True)

    tokenized_datasets = raw_datasets.map(preprocess_function, batched=True)
    tokenized_datasets = tokenized_datasets.remove_columns(["sentence1", "sentence2",
"idx"])
    tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
    tokenized_datasets.set_format("torch")

    # Run the ensemble function
    ensemble_set = ensemble_bert(
        num_iterations=wandb_config["num_iterations"],
        learning_rate=wandb_config["learning_rate"],
        batch_size=wandb_config["batch_size"],
        convergence_threshold=wandb_config["convergence_threshold"],
    )

    num_tickets = len(ensemble_set)
```

```
  print("Number of tickets (subnets) in the ensemble:", num_tickets)

  # Plotting Number of Tickets vs. Ensemble Iteration
  plt.plot(range(len(ensemble_set)), [num_tickets] * len(ensemble_set), label="Number of
Tickets")
  plt.title("Number of Tickets vs. Ensemble Iteration")
  plt.xlabel("Ensemble Iteration")
  plt.ylabel("Number of Tickets")
  plt.legend()
  plt.show()

  # Evaluate the ensemble and log accuracy
  accuracy_scores = []
  for iteration, model in enumerate(ensemble_set):
    trainer = Trainer(
      model=model,
      args=training_args,
      train_dataset=tokenized_datasets["train"],
      eval_dataset=tokenized_datasets["validation"],
      data_collator=data_collator,
      tokenizer=tokenizer,
      compute_metrics=compute_metrics,
    )
    trainer.train()
    predictions = trainer.predict(tokenized_datasets["test"])
    accuracy = accuracy_score(predictions.label_ids, predictions.predictions.argmax(-1))
    accuracy_scores.append(accuracy)

    # Log accuracy to wandb
    wandb.log({"ensemble_iteration": iteration, "accuracy": accuracy})

  # Save the ensemble models
  os.makedirs("ensemble_models", exist_ok=True)
  for i, model in enumerate(ensemble_set):
    model.save_pretrained(f"ensemble_models/model_{i}")

  # Save the best model to the Hugging Face Hub
  best_model_index = accuracy_scores.index(max(accuracy_scores))
  best_model = ensemble_set[best_model_index]
  model_name = "your_hub_username/your_model_name"
  best_model.save_pretrained(model_name)
  tokenizer.save_pretrained(model_name)
  wandb.save(f"ensemble_models/model_{best_model_index}/config.json")
```

```
# Plot Accuracy vs. Ensemble Iteration
plt.plot(range(len(accuracy_scores)), accuracy_scores)
plt.title("Accuracy vs. Ensemble Iteration")
plt.xlabel("Ensemble Iteration")
plt.ylabel("Accuracy")
plt.show()

# Finish wandb run
wandb.finish()
```