

# IMPROVING THE PERFORMANCE OF QUANTIZED TRANSFORMERS WITH GRAPH NEURAL NETWORKS

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE  
OF BILKENT UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR  
THE DEGREE OF  
MASTER OF SCIENCE  
IN  
ELECTRICAL AND ELECTRONICS ENGINEERING

By  
Cihan Eralp Kumbasar  
May 2025

Improving the Performance of Quantized Transformers with Graph  
Neural Networks

By Cihan Eralp Kumbasar

May 2025

We certify that we have read this thesis and that in our opinion it is fully adequate,  
in scope and in quality, as a thesis for the degree of Master of Science.

---

Aykut Koç(Advisor)

---

Haldun M. Özaktaş(Co-Advisor)

---

Tolga Çukur

---

Elif Vural

Approved for the Graduate School of Engineering and Science:

---

Orhan Arıkan  
Director of the Graduate School

# ABSTRACT

## IMPROVING THE PERFORMANCE OF QUANTIZED TRANSFORMERS WITH GRAPH NEURAL NETWORKS

Cihan Eralp Kumbasar

M.S. in Electrical and Electronics Engineering

Advisor: Aykut Koç

Co-Advisor: Haldun M. Özaktas

May 2025

Transformers have become established models in natural language processing (NLP) tasks. Their representational capabilities improve with size, but training and hosting larger models is computationally demanding. The rise in computational overhead leads to an increase in the carbon footprint, raising concerns about the environmental impacts of using these models. Parameter quantization promises to reduce the utilization costs of transformers; however, low-bit quantization generally leads to a notable loss in model performance. To reduce the degradation in model performance due to quantization while maintaining its associated benefits, we introduce BitTransGNN, a novel framework that improves quantized transformer performance by integrating them with Graph Neural Networks (GNNs). Transformers excel in capturing local contextual semantics, while GNNs are competent in representing global structural relationships within data. BitTransGNN makes use of this complementary nature of the two models to improve the representational capabilities of quantized transformers. After presenting our proposed architecture, to extend the utility of BitTransGNN to inductive settings, we then introduce variants of BitTransGNN that encapsulate the knowledge learned by BitTransGNN within a solitary quantized transformer model. Through an extensive set of experiments, we show that BitTransGNN substantially reduces the performance gap between quantized transformers and their full-precision counterparts while retaining the efficiency advantages provided by quantization. Transductive BitTransGNN variants outperform quantized transformer baselines by up to 21% while introducing minimal additional overhead. Inductive BitTransGNN variants improve quantized transformer performance by up to 19% with zero additional inference costs. To evaluate the cost-performance tradeoff, we inspect the model performance and utilization costs of BitTransGNN

and the baseline models. We perform further analyses on BitTransGNN outputs to validate the premise that transformers and GNNs focus on highly distinct features, examine the significance of different BitTransGNN components, and discuss potential limitations. The results and findings presented in this thesis contribute to the research on improving the efficiency of neural networks and offer a new perspective on reducing neural model costs without making important sacrifices from model performance.



*Keywords:* Transformer, Quantization, Binary Neural Networks, Quantized Neural Networks, Efficient Transformers, Large Language Models (LLMs), Graph Neural Network (GNN), Knowledge Distillation.

# ÖZET

## NİCEMLENMİŞ DÖNÜŞTÜRÜCÜLERİN BAŞARIMININ GRAFİK SİNİR AĞLARI İLE GELİŞTİRİLMESİ

Cihan Eralp Kumbasar  
Elektrik ve Elektronik Mühendisliği, Yüksek Lisans  
Tez Danışmanı: Aykut Koç  
İkinci Tez Danışmanı: Haldun M. Özaktaş  
Mayıs 2025

Dönüştürücüler, Doğal Dil İşleme (NLP) görevlerinde yerleşik modeller haline gelmiştir. Dönüştürücülerin temsil yetenekleri model boyutuyla doğru orantılı gelişmektedir; ancak, büyük modelleri eğitmek ve barındırmak yoğun bir işlemci kaynağı gerektirmektedir. Artan işlemci maliyeti, karbon ayak izinin yükselmesine yol açmaktadır. Bu durum, modellerin kullanımının yol açacağı çevresel etkilere dair endişeler oluşturmaktadır. Model değişkenlerinin nicemlenmesi, dönüştürücülerin kullanım maliyetinin azaltılması açısından umut vadetmektedir; ancak, düşük-bit temsillere nicemleme genellikle model başarımında önemli bir düşüşe sebep olmaktadır. Nicemlemenin model başarımında sebep olduğu düşüşü, nicemlemenin sağladığı faydaları koruyarak sınırlamak için BitTransGNN modelini sunuyoruz. BitTransGNN, nicemlenmiş dönüştürücülerini Çizge Sinir Ağları (GNN'ler) ile entegre ederek bu modellerin performansını geliştirmeyi amaçlayan özgün bir mimaridir. Dönüştürücüler yerel bağlamsal bilgiyi modellemekte büyük başarı gösterirken, GNN'ler küresel yapısal ilişkileri temsil etme kabiliyetleriyle öne çıkmaktadırlar. BitTransGNN, nicemlenmiş dönüştürücülerin temsil kabiliyetlerini geliştirmek için iki modelin birbirini tamamlayan bu yapılarından faydalanır. Model mimarimizi tanıttıktan sonra, BitTransGNN'in kullanımını indüktif problem koşullarına genişletmek için BitTransGNN'in öğrendiği bilgiyi tekil bir nicemlenmiş dönüştürücü modeline aktaran modeller ortaya koyuyoruz. Kapsamlı deneylerle, BitTransGNN'in, nicemlemenin sunduğu verimlilik avantajlarını koruyarak, nicemlenmiş dönüştürücüler ve onların tam hassasiyetli eşlerinin arasındaki başarımlarını önemli ölçüde azalttığını gösteriyoruz. Transdüktif BitTransGNN modelleri, yalnızca az miktarda ek maliyetle, nicelenmiş dönüştürücü temel modellerden %21 oranına

kadar daha iyi başarımlar göstermektedir. İndüktif BitTransGNN modelleri, ek bir çıkarsama maliyetine yol açmadan, nicemlenmiş dönüştürücü başarımlarını %19 oranına kadar arttırmaktadır. Maliyet-başarımlar dengesini, BitTransGNN ve temel modellerin kullanım maliyetlerini ve başarımlarını inceleyerek değerlendiriyoruz. Dönüştürücü ve GNN'lerin oldukça farklı özelliklere odaklandığı öncülünü doğrulamak için ek analizler yapıyor, farklı BitTransGNN bileşenlerinin önemini inceliyor ve modelin olası limitlerini tartışıyoruz. Bu tezde sunulan sonuçlar ve bulgular, sinir ağlarının verimliliğinin artırılması üzerine yapılan araştırmalara katkıda bulunmakta ve sinir ağı temelli modellerin kullanım maliyetinin, model başarımlarında önemli kayıplar verilmeden geliştirilmesi üzerine yeni bir bakış açısı sunmaktadır.

*Anahtar sözcükler:* Dönüştürücü, Nicemleme, İkili Sinir Ağları, Nicemlenmiş Sinir Ağları, Verimli Dönüştürücüler, Büyük Dil Modelleri (LLM'ler), Çizge Sinir Ağı (GNN), Bilgi Damıtımı.

## Acknowledgement

*This work is supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK), Directorate of Science Fellowships and Grant Programmes (BİDEB), through the 2210-A National MSc Scholarship Program and partially by the Directorate of Research Support Programs (ARDEB) 1001-120E346.*

I would like to express my gratitude to my supervisor Asst. Prof. Aykut Koç for his guidance during my Master's studies and through the construction of this thesis. Working as part of his group has given me valuable lessons that I will benefit from throughout my academic and professional life. I would also like to thank my co-advisor Prof. Haldun Özaktas for his mentorship and support. I have greatly benefited from the valuable feedback that he has provided and his guidance has greatly contributed to my research. I would also like to thank the members of the thesis committee, Prof. Tolga Çukur and Assoc. Prof. Elif Vural, for their time and interest.

I would like to thank the members of our research group, Ecem Şimşek, Emirhan Koç, Tuna Alikasıfoğlu, Arda Can Aras and Tuna Özateş, for their help and support, and for enriching this experience with their presence. Having the opportunity to collaborate with you has brought value to the time I have worked as part of this group.

Finally, I would like to express my deepest gratitude to my family. To my parents, Dr. Özlem Kumbasar and Dr. Hakan Kumbasar, thank you for the endless support and infinite patience you have shown and for always believing me, especially at the times when I doubted myself the most. To my brother Erdem, thank you for showing tremendous emotional support throughout this process. You have set a great example for me with a remarkable ability to handle stressful situations. Hopefully, I will be able to learn more from you next year when we are not separated by continents. I would like to thank each member of my family for being the source of happiness and laughter in my life and doing whatever they can to help me whenever I needed their assistance. I would also

like to thank each and every one of my friends for supporting me throughout the entire journey and bringing me joy.



# Contents

|          |                                                                |           |
|----------|----------------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                            | <b>1</b>  |
| 1.1      | Utilization Costs of Transformers . . . . .                    | 1         |
| 1.2      | Neural Network Quantization . . . . .                          | 2         |
| 1.3      | Motivation . . . . .                                           | 3         |
| 1.4      | Contributions . . . . .                                        | 4         |
| 1.5      | Organization . . . . .                                         | 7         |
| <b>2</b> | <b>Related Work</b>                                            | <b>8</b>  |
| 2.1      | Quantization in Neural Networks . . . . .                      | 8         |
| 2.2      | Graph Neural Networks . . . . .                                | 9         |
| 2.3      | Graph Neural Networks in Natural Language Processing . . . . . | 11        |
| 2.4      | Knowledge Distillation over Transformer Networks . . . . .     | 12        |
| <b>3</b> | <b>BitTransGNN Architecture</b>                                | <b>14</b> |
| 3.1      | BitTransformer Module . . . . .                                | 16        |

|          |                                                         |           |
|----------|---------------------------------------------------------|-----------|
| 3.1.1    | Quantization Procedure . . . . .                        | 16        |
| 3.1.2    | Quantized Transformer Blocks . . . . .                  | 18        |
| 3.1.3    | Integrating Quantization and Fine-Tuning . . . . .      | 20        |
| 3.2      | GNN Module . . . . .                                    | 21        |
| 3.2.1    | Graph Generation . . . . .                              | 22        |
| 3.2.2    | GNN Blocks . . . . .                                    | 23        |
| 3.3      | The Integration Module . . . . .                        | 24        |
| 3.4      | Training Schemes . . . . .                              | 25        |
| 3.4.1    | BitTransGNN <sub>Static</sub> . . . . .                 | 25        |
| 3.4.2    | BitTransGNN <sub>Dynamic</sub> . . . . .                | 26        |
| 3.5      | Inductive BitTransGNN Variants . . . . .                | 26        |
| 3.5.1    | Direct Separation (DS) . . . . .                        | 27        |
| 3.5.2    | Induction Through Knowledge Distillation (KD) . . . . . | 28        |
| <b>4</b> | <b>Efficiency Analysis</b>                              | <b>31</b> |
| 4.1      | Memory Efficiency . . . . .                             | 31        |
| 4.2      | Energy Efficiency . . . . .                             | 33        |
| <b>5</b> | <b>Experimental Results and Evaluations</b>             | <b>40</b> |
| 5.1      | Datasets . . . . .                                      | 40        |

|          |                                                             |           |
|----------|-------------------------------------------------------------|-----------|
| 5.2      | Hyperparameter Search . . . . .                             | 42        |
| 5.3      | Experimental Setup . . . . .                                | 43        |
| 5.4      | Results and Evaluations . . . . .                           | 44        |
| 5.4.1    | Transductive Model Performance . . . . .                    | 45        |
| 5.4.2    | Inductive Model Performance . . . . .                       | 45        |
| 5.4.3    | Efficiency-Performance Analysis . . . . .                   | 47        |
| 5.4.4    | T-SNE Plots . . . . .                                       | 51        |
| <b>6</b> | <b>Discussion</b>                                           | <b>54</b> |
| 6.1      | The significance of $\lambda$ . . . . .                     | 54        |
| 6.2      | The Synergy Between BitTransformer and GNN Models . . . . . | 57        |
| 6.3      | Choice of Quantization Configuration . . . . .              | 58        |
| 6.4      | Limitations and Drawbacks of BitTransGNN Models . . . . .   | 62        |
| <b>7</b> | <b>Conclusion</b>                                           | <b>64</b> |
| <b>A</b> | <b>Source Code</b>                                          | <b>80</b> |

# List of Figures

|     |                                                                                                                                                                                                                                                                                                                     |    |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1 | Proposed BitTransGNN architecture, including transformer quantization and fine-tuning, output production of BitTransformer and GNN modules, and the integration of the two models. The schematic contains a BitTransGNN model with an L-layered Bit-Transformer and a 2-layered GNN that employs graph convolution. | 16 |
| 3.2 | The computation of the knowledge distillation loss. . . . .                                                                                                                                                                                                                                                         | 29 |
| 5.1 | Performance of BitTransGNN variants and baseline methods against memory consumption (left) and enery consumption (right).                                                                                                                                                                                           | 50 |
| 5.2 | Performance of BitTransGNN and baseline models against energy consumption when int32 (left) and int8 (right) data types are utilized.                                                                                                                                                                               | 52 |
| 5.3 | T-SNE embeddings of BitTransGNN <sub>Dynamic</sub> (left) and BitTransGNN-KD (right) outputs on the 20NG dataset displayed in a two dimensional space. Colors represent different classes predicted by the models. .                                                                                                | 53 |
| 6.1 | BERT-based BitTransGNN <sub>Static</sub> model performance against vary-<br>ing values of $\lambda$ listed for different quantization rates. . . . .                                                                                                                                                                | 56 |

6.2 PCA analysis on the similarity profiles of the logits of BitTransformer and GNN modules for the BitTransGNN<sub>Dynamic</sub> model trained on the 20NG dataset. Similarity profiles are listed for BitTransGNN<sub>Dynamic</sub> models quantized to different rates. The importance profiles are computed using RSA. Circles correspond to the components from the BitTransformer, and squares correspond to the components from the GNN models of BitTransGNN. Colors represent different classes that the model distinguishes during classification. . . . . 63

# List of Tables

|     |                                                                                                                                                                                                                                           |    |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1 | Summary of Notation. . . . .                                                                                                                                                                                                              | 15 |
| 3.2 | Sets used to quantize the full-precision weights. . . . .                                                                                                                                                                                 | 17 |
| 3.3 | Performance of quantized models when Quantization-Aware Fine-Tuning (QAFT) and Post Fine-Tuning Quantization (PFTQ) are used as fine-tuning strategies. . . . .                                                                           | 22 |
| 4.1 | Exact sizes of BERT and RoBERTa transformer components. . .                                                                                                                                                                               | 32 |
| 4.2 | Number of parameters and the number of bits allocated for parameters contained within different BERT transformer components and the convolutional GNN model. We generally use the 20NG model as reference, which has $d_o = 20$ . . . . . | 32 |
| 4.3 | Number of bits required to store BitTransformer and BitTrans-GNN parameters in different quantization scenarios. . . . .                                                                                                                  | 34 |
| 4.4 | Number of addition and multiplication operations performed by the transformer. Exact operation counts for different model variants can be calculated using the variables listed in Table 4.1. We use a window length $M = 128$ . . . . .  | 36 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |    |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 4.5 | Count of additions and multiplications, and the ratio of arithmetic operations for BitTransformer models against their full-precision counterparts. . . . .                                                                                                                                                                                                                                                                                                                 | 37 |
| 4.6 | Unit energy consumption of addition and multiplication operations listed for 7 nm and 45 nm processors, and the energy ratio between the two operations for three different data types. . . . .                                                                                                                                                                                                                                                                             | 37 |
| 4.7 | Energy consumption by arithmetic operations per epoch for baseline quantized transformer models and BitTransGNN variants over 7nm processors. Ratios of the energy overhead of each instance against that of the full-precision transformer baseline are also listed alongside the exact values of energy expense. . . . .                                                                                                                                                  | 39 |
| 5.1 | Statistics and information related to datasets. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                     | 42 |
| 5.2 | Parameter sets that are swept during hyperparameter search. . . .                                                                                                                                                                                                                                                                                                                                                                                                           | 43 |
| 5.3 | Comparison of BitTransGNN methods with quantized and full-precision transformers. The best results are boldened, the second-best models are underlined, and the third-best methods are italicized.                                                                                                                                                                                                                                                                          | 46 |
| 5.4 | Comparison of inductive BitTransGNN variants against full-precision and quantized transformer baselines. The best results are boldened, the second-best models are underlined, and the third-best methods are italicized. . . . .                                                                                                                                                                                                                                           | 48 |
| 6.1 | Study to examine the effect of quantizing different parts of the BERT model on model performance. $W\{b_l\}E\{b_e\}A\{b_a\}$ refers to a quantization instance of the transformer variant where the embedding weights are quantized to $b_e$ bits, the parameters in the rest of the modules are quantized to $b_l$ bits, and the activations right before these parameters are quantized to $b_a$ bits. BitTransformer models fine-tuned with QAFT are considered. . . . . | 59 |

- 6.2 Memory consumption of BitTransformer model parameters in different binarization scenarios, listed for different BERT and RoBERTa model sizes.  $W\{b_l\}E\{b_e\}$  refers to embedding weights quantized to  $b_e$  bits and rest of the parameters quantized to  $b_l$  bits. 61



# Disclaimer

This thesis includes content from the following publication:

- C. E. Kumbasar, H. M. Özaktas, and A. Koç, “Graph neural networks improve quantized transformers by incorporating global structural relationships,” *IEEE Transactions on Signal and Information Processing over Networks*, 2025. Manuscript under review. [\[1\]](#)

# Chapter 1

## Introduction

In this chapter, we first highlight the considerable computational and financial burden of hosting large-scale transformer architectures. Then, we outline the use of quantization strategies on neural networks to reduce the utilization costs of neural models. We then present the motivation of our study to reduce the performance loss caused by quantization while pursuing an improvement in the efficiency of transformer models. We later list the contributions of our work and outline the organization of this thesis.

### 1.1 Utilization Costs of Transformers

Transformers display remarkable success in many natural language processing (NLP) tasks, such as machine translation, named entity recognition, and language modeling [2–5]. The exceptional representational capabilities of transformers have also received attention from other domains, including image, video, audio, speech, and time-series processing [6–12].

The performance of neural models improves with increasing model size [13]. This pattern led to the use of larger transformers in language processing tasks,

which eventually led to the introduction of Large Language Models (LLMs) containing highly parameterized transformer networks at their cores [14]. Although transformers display unmatched performance, training and hosting larger transformer models is challenging due to increased computational costs [15], limiting the accessibility of these models. Even when the financial necessities are met, the use of computationally demanding models increases energy consumption and carbon footprint, raising environmental concerns regarding the use of such models [16]. As an illustrative example, the entire training process of the Generative Pre-Trained Transformer 3 (GPT-3) [17] model containing 175 billion parameters produced a carbon footprint of over 552,100 kilograms of CO<sub>2</sub>-equivalents [18]. This footprint is equivalent to more than 110 times the average yearly CO<sub>2</sub> emission produced by a human being [19]. Moreover, interest in hosting transformer-based models on smaller devices, such as mobile phones, further motivates the reduction of computational costs [20].

## 1.2 Neural Network Quantization

Quantization is a promising tool to improve the inference efficiency of transformers. Quantization was instrumental in improving the efficiency of early deep learning models by compressing full-precision parameters to low-bit precision [21–26]. The extreme case of quantization is binarization, which maps full-precision parameters to the set  $\{-1, 1\}$ . Such mapping offers to reduce memory costs by storing model parameters with single bits and decrease computational costs by replacing resource-intensive matrix multiplication operations with more cost-efficient addition and subtraction operations. Transformer models have greatly benefited from quantization. Quantized versions of different transformer models have been introduced to reduce the computation and memory overhead of these models [27–32].

Despite promising to improve efficiency, binarization leads to a significant drop in the performance of transformer models [28, 29]. Some methods use relatively higher precision during quantization to reduce this performance loss (e.g., 8-bits or 16-bits) [31] while making significant sacrifices from potential efficiency

advantages. Others quantize model weights to low-bit precision after training for efficient use during inference time, but lose most of the information learned by the full-precision model [20]. Training using the binary parameter set produces better results [20, 33]; however, the methods still underperform compared to their full-precision counterparts due to reduced representational capabilities.

### 1.3 Motivation

Recognizing the substantial costs required to host large transformer models and the inadequacy of neural network quantization strategies in maintaining the performance of full-precision transformers, this thesis aims to improve quantized transformer performance while preserving the efficiency advantages provided by quantization. In that regard, we introduce BitTransGNN, a novel framework that utilizes Graph Neural Networks (GNNs) [34–41] to improve quantized transformer training. BitTransGNN reduces the performance gap between transformers quantized to low-bit precision and their full-precision counterparts with minimal additional overhead by leveraging GNNs.

Transformers’ success is based on its ability to grasp local contextual information within sequences through the use of a finite-context self-attention window [2]. Despite successfully representing the local context, self-attention is incapable of processing any information contained outside its window. Increasing the span of the window is not a feasible solution due to increased costs, as the time-complexity of self-attention scales quadratically with respect to sequence length. This limits transformers’ ability to capture global relationships across data entities.

This shortcoming of transformer models forms the key motivation of this thesis. We improve the knowledge representation of quantized transformers by addressing their limitations in capturing long-range dependencies. Using GNNs, we compensate for the performance loss arising from quantization by providing quantized transformers with information on the global structural relationships across data entities. Transformers show remarkable proficiency in modeling local contextual

information, whereas GNNs are highly effective in modeling global structural information within the input data. The complementary nature of the two models allows us to overcome the trade-off between cost and accuracy that comes with quantization by collaborating a quantized transformer with a GNN. To this end, we represent the relationships across data entities through a graph structure and propose a GNN-based method to process the graph-structured data. We employ graph convolutions in our GNN module, which update the state of each graph node by aggregating information through its neighborhood [34, 36, 40–42]. Graph Convolutional Networks (GCNs) integrate graph convolution into GNN models to learn node features [34]. GCN has linear complexity with respect to the number of graph edges and can exploit the sparsity within the graph structure to reduce the number of computations it performs. This allows GCN to operate while consuming significantly less energy than transformers. In addition, storing GNN parameters requires notably less memory than transformers, making them more efficient in terms of parameter memory costs as well. We improve quantized transformer performance through an efficient procedure by leveraging these advantages of GNNs and graph convolution.

## 1.4 Contributions

To overcome the deficiencies of existing methods in restricting the performance loss induced by quantization, we introduce BitTransGNN, a framework that jointly trains a quantized transformer with a GNN model in different configurations. BitTransGNN integrates the quantized transformer and the GNN model in the training stage to improve the representational capabilities of the quantized transformer model. The joint architecture improves quantized transformer training while introducing minor additional costs influenced by the incorporation of GNN parameters.

Having presented the motivations underlying the construction of BitTransGNN and the foundations of the model architecture, we then focus on extending our model’s utility to transductive and inductive problem settings [43]. Inductive

learning deduces rules about the data distribution and uses these rules to make inference on unseen instances [39]. Contrarily, transductive learning does not construct a general rule and directly derives reasoning from data samples for specific test cases [44]. Inductive training only learns from training instances, whereas transductive training can also learn from the patterns within the test data. Both methods offer unique advantages and disadvantages, and the choice of the method depends on the problem context.

BitTransGNN is inherently transductive due to the direct involvement of the GNN on the final output. Despite its unique advantages, the transductive characteristic of BitTransGNN contradicts the inductive nature of transformer models, creating a difficulty in comparing BitTransGNN models with transformer variants. To make BitTransGNN compatible with inductive problem settings, we introduce two methods that extend the use of BitTransGNN to inductive tasks by encapsulating the knowledge learned by a BitTransGNN model during training into a quantized transformer. The first method separates the quantized transformer from BitTransGNN and uses it for inference without any additional training. The second method introduces a low-cost knowledge distillation framework [45]. Distillation was originally introduced to transfer knowledge from a larger, parameter-heavy teacher model into a smaller, less-parameterized student [45–50]. We use distillation to compress the knowledge learned by BitTransGNN into a solitary quantized transformer. Through distillation, quantized transformer performance is improved by training on BitTransGNN outputs as well as on ground-truth labels. Both methods remove quantized transformer’s dependence on the GNN model after BitTransGNN training, extend BitTransGNN’s utility to inductive problem settings, and maintain the efficiency advantages introduced by quantization during inference time.

BitTransGNN models outperform quantized transformers fine-tuned over various language tasks with only a minor additional overhead. BitTransGNN architecture generally shows competitive performance against full-precision transformers, despite having reduced representational capabilities due to quantization. By synergistically using the quantized transformer and the GNN, we make use of the complementary way in which transformers and GNNs capture local and global

relationships, and attain a cost-performance point superior to what is achieved by training transformers in an isolated manner. In addition to the achievements of the transductive BitTransGNN methods, the proposed inductive BitTransGNN models effectively compress the knowledge learned by BitTransGNN within a solitary quantized transformer. These variants make it possible for us to adapt the BitTransGNN framework for inductive inference tasks. Inductive BitTransGNN models improve quantized transformer performance with zero additional inference costs and show competitive results against full-precision transformers, even without GNN’s assistance during inference. We show that quantization can theoretically reduce the memory required to store transformer parameters by 4 to 32 times and that the BitTransGNN architecture imposes only a very slight additional memory overhead compared to quantized transformer models. We also theoretically demonstrate that quantization can halve the energy consumption of transformers’ arithmetic operations and that BitTransGNN variants can exploit this advantage with only small additional costs. We elaborate to illustrate that the expenditure can be reduced even further if the network processes information through more efficient data types. We perform additional analyses over BitTransGNN methods to analyze our premise on the complementary nature of quantized transformers and GNNs, provide further insight into the significance of different components within our framework, and to emphasize the contributions of our method.

We can summarize the contributions of this thesis as follows:

- We propose BitTransGNN, an architecture that substantially reduce, and generally closes, the performance gap between quantized transformers and their full-precision counterparts while retaining the efficiency advantages associated with quantization.
- We introduce BitTransGNN that encapsulate the knowledge learned by the joint model into an independent quantized transformer to extend the utility of our architecture to inductive problem contexts.
- We show that BitTransGNN prevents to a substantial degree the performance loss that would otherwise accompany a major reduction in memory

and energy consumption. We theoretically demonstrate a 4- to 32-fold reduction in parameter memory usage and a 2-fold decrease in energy consumption enabled by quantization.

- BitTransGNN outperforms quantized transformers by as much as 21% with minor additional memory overhead and shows competitive performance against full-precision variants.
- The inductive BitTransGNN methods improve quantized transformer performance by up to 19% with zero additional inference costs.

## 1.5 Organization

The rest of this thesis is organized as follows. We first review the related work on quantized transformers, GNNs and their applications in NLP, and the use of knowledge distillation on transformer networks in Chapter 2. Then, we present the BitTransGNN framework in Chapter 3. We show the theoretical efficiency gains that our architecture can achieve through transformer quantization in Chapter 4, where we discuss potential improvements in terms of both memory requirements and energy consumption. In Chapter 5, we outline the details of our experiments and demonstrate our results. We present discussions to assess the significance of our findings in Chapter 6. Finally, we conclude in Chapter 7.

# Chapter 2

## Related Work

This chapter reviews some of the related work on the study being presented. We first cover works on quantization and binarization in neural networks and transformers. Then, we present the foundations of GNNs. Later, we explore the use of GNNs in NLP applications. Lastly, we present several works that reduce transformer utilization costs and improve quantized transformer performance through knowledge distillation.

### 2.1 Quantization in Neural Networks

Quantization and binarization strategies were first adopted to improve the efficiency of early deep learning approaches. [23] and [21] train a network with binary weights and activations while maintaining the gradients high precision for the stochastic gradient descent (SGD) to operate. With specialized kernels, they show improvements in the inference efficiency of neural networks. Binarization maps real-valued neural network weights to the set  $\{-1, 1\}$ . This allows to replace computationally demanding multiplication operations with resource-friendly additions and subtractions. The ternary version that maps real-valued parameters to the set  $\{-1, 0, 1\}$  also shows this promise [24, 51]. Alternative sets, such as

$\{-2, -1, 0, 1, 2\}$ , can also provide this advantage, as multiplication can also be eliminated in this scenario by using shift operations in binary arithmetic [25, 26].

Due to their substantial computational demands, transformers greatly benefit from quantization. Several works propose quantized alternatives to different transformer variants. Q8BERT [31] proposes to reduce BERT memory consumption to a quarter of its original amount by quantizing model weights to 8-bit integers. Studies such as BEBERT [27], BiBERT [28] and BinaryBERT [29] propose different methods to train binary versions of the BERT transformer. Apart from BERT variants, BiViT introduces a binary vision transformer model [30]. LLMs can also benefit from quantization, as they have a higher carbon footprint due to larger scaling [52]. Aiming to address these concerns, [20] and [33] train binary and ternary LLM variants from scratch to significantly reduce the inference costs of LLMs.

## 2.2 Graph Neural Networks

The modeling of data on irregular structures, such as graphs and networks, has gained increasing attention in recent years, as these representations effectively capture complex interactions between data entities. Graph Signal Processing (GSP) emerged as a powerful framework for processing signals and information through graph structures [53–57]. Following the introduction of GSP, many fundamental signal processing concepts defined in the Euclidean domain have been extended to the irregular graph domain [58–66]. The concepts developed under the GSP paradigm have been applied to a wide range of fields and application domains [67–72].

Building on the framework established by GSP, machine learning methods have been proposed to learn representations and perform predictive tasks on graph structures, eventually leading to the introduction of Graph Neural Networks (GNNs) [34–41]. GNN models extend neural network architectures to

non-Euclidean domains, allowing the processing of data not confined to a regular grid structure. One of the principal operations over graph-structured data is graph convolution, which operates through the graph neighborhood to aggregate information over graph nodes. Initial efforts focused on extending the concept of convolution to the graph domain by transferring principles from the Convolutional Neural Network (CNN) architecture and adapting them to GNN models [35, 36, 42]. Expanding on these works, [34] proposed GCN, a pioneering model in the GNN literature that learns node features by propagating information through graph nodes using graph convolution. GCN uses a localized first-order approximation of the spectral graph convolution and offers a scalable model that grows linearly with increasing number of graph nodes. The model achieved state-of-the-art results in node classification tasks. Each GCN layer averages the hidden representations of the nodes’ immediate neighbors, i.e., the 1-hop neighborhood, so that after  $k$ -stacked layers, each node obtains feature information from nodes that are  $k$ -hops away. [40] introduces further simplifications to the GCN model by removing the non-linear activation functions between consecutive layers.

Despite its effectiveness, GCN is inherently transductive. It learns based on the entire graph structure supplied to the model. It cannot generalize to unseen node instances and has to be re-trained in case new nodes are added to the graph. To address this limitation, GraphSAGE proposed an inductive framework that samples and aggregates features from each node’s neighborhood to learn a function that can generate embeddings for unobserved nodes [39].

The computational complexity of graph convolution scales with  $\mathcal{O}(E \cdot d)$ , where  $E$  is the number of edges between graph nodes. Self-attention mechanism in transformer models processes the relations between each entity over a fully-connected graph structure that contains the tokens within the sequence at its nodes. In such scenario, since all nodes are connected to each other,  $E = N^2$ , where  $N$  is the number of nodes within the graph. Accordingly, the model complexity of the transformer scales quadratically with respect to sequence length. In contrast, GNNs usually operate over sparse graph structures, where  $E \ll N^2$ , and their complexity scales linearly with respect to the number of edges. As a consequence, they do not require as much memory to store model parameters compared

to transformer models.

## 2.3 Graph Neural Networks in Natural Language Processing

Earlier developments in NLP were driven by recurrent methods, such as Recurrent Neural Network (RNN) [73, 74], Long-Short Term Memory (LSTM) [75] and Gated Recurrent Unit (GRU) [76], as well as methods that employ gated convolutions [77], to capture the local context within sequences. These models were able to handle long sequences with sub-quadratic time complexity; however, they lacked a selection mechanism to detect important information while handling long-range dependencies [78]. Due to the success of its attention mechanism in capturing intricate relationships within its window, transformers became the state-of-the-art method in NLP despite their quadratic time-complexity [2]. In addition to the success of its attention mechanism, transformers also benefit from large-scale pre-training to learn from vast amounts of data, which allows them to generalize better to different tasks [3, 4]. Recognizing the potential of the architecture, numerous studies explored modifications to the transformer model to further enhance its performance in language tasks [2–5, 79–81]. A supporting approach to the sequential processing of text data is to model the entire corpus through graph structures to capture the global context and represent relationships that are beyond the scope of the local context. GNNs appeared as the prevailing method for processing information through text graphs. Due to the success of this approach, GNN-based methods have gained popularity in a variety of NLP tasks [82], including text classification [80, 83–86], neural machine translation [87, 88], and text summarization [89, 90].

TextGCN [83], a pioneering work that utilizes GNNs in NLP tasks, models text classification through a graph perspective. The authors placed the words and documents within a corpus at the nodes of a text graph. The relationships between graph nodes were determined through text statistics. The term

frequency-inverse document frequency (TF-IDF) metric is used to determine word co-occurrence within documents, while the Positive Point-wise Mutual Information (PPMI) statistic is used to measure word co-occurrence across the whole corpus. TextGCN uses a homogeneous GNN to propagate information through the heterogeneous text graph, updating word and document embeddings identically [83]. In heterogeneous graphs, relationships between different node types can be partitioned with some additional overhead to improve their feature representations, and the data can be processed using heterogeneous GNNs to make use of the diverse patterns enabled by the heterogeneity [84, 91]. HeteGCN [84] exploits the heterogeneous structure of TextGCN’s text graph by proposing different message propagation paths between word and document nodes. In addition to the direct use of GNNs in NLP tasks, several works collectively benefit from the advantages of transformers and GNNs. BertGCN [85] jointly trains a fine-tuned BERT model with a GCN. Graph Receptive Transformer Encoder (GRTE) [86] integrates the document embeddings produced by BERT to the document nodes of the text graph and propagates information using the paths proposed in HeteGCN. A similar collaboration was also established between LLMs and GNNs in different works to enhance the performance of both systems [92–97].

## 2.4 Knowledge Distillation over Transformer Networks

It has been demonstrated in [46] that it is possible to compress the knowledge contained in an ensemble of models into a single isolated model. Generalizing on this proposition, [45] proposed knowledge distillation, a method that transfers knowledge from a larger, parameter-heavy teacher model into a smaller student containing less parameters. Distillation improves student model performance by jointly training it on the teacher model outputs as well as the ground-truth labels.

Due to their intensive computational needs, transformers greatly benefit from

knowledge distillation. DistilBert [47], TinyBert [49], and MiniLM [48] use distillation to reduce the size of BERT-based language models. DistilBert employs distillation in the pre-training stage of BERT to reduce model size by 40% while retaining most of its capabilities [47]. MiniLM compresses the knowledge learned by large transformer into a smaller student model by training the student’s self-attention module to mimic the self-attention mechanism of the teacher. The method distills knowledge from the final self-attention layer of the teacher into the student network [48]. TinyBERT conducts distillation across transformer models in both pre-training and fine-tuning stages [49].

Inspired by the method’s success in model compression, many works used distillation as a means for transferring the capabilities of full-precision transformers into quantized models to improve quantized model performance while retaining the efficiency advantages offered by quantization. BiT transfers knowledge from models with higher-precision weights into a binary transformer to bring its performance closer to full-precision transformer variants [50]. BEBERT [27] integrates binary and full-precision BERT models through ensemble learning. The authors explore different distillation approaches to train these ensembles and improve binary BERT performance. In [28], the authors claim that binary BERT training with distillation can suffer due to a direction mismatch during backpropagation. To address this issue, they propose a Direction Matching Distillation scheme to train a binary BERT.

## Chapter 3

# BitTransGNN Architecture

This chapter presents the proposed BitTransGNN architecture in detail. A visual representation of BitTransGNN is presented in Fig. 3.1. We generally operate BitTransGNN on sequence-level tasks, which correspond to document-level tasks in the NLP context. We will use the two terms interchangeably throughout this chapter. We refer to transformer instances that we quantize to low-bit precision as BitTransformer in our study and as BitTrans in equations for brevity. We first explain the transformer quantization procedure, the information processing of the quantized transformer blocks, and how we incorporate quantization into the transformer fine-tuning stage in Chapter 3.1. We outline the generation of the text-graph and how the GNN module processes the graph-structured data in Chapter 3.2. We formulate how BitTransGNN is built by integrating the BitTransformer and GNN modules in Chapter 3.3. We finally introduce the methods used to train BitTransGNN in Chapter 3.4.

A list of the notations of the variables used in our formulation can be found in Table 3.1.

Table 3.1: Summary of Notation.

| Notation                                                             | Meaning                                     |
|----------------------------------------------------------------------|---------------------------------------------|
| $N$                                                                  | Number of sequences                         |
| $T$                                                                  | Total number of token types within corpora  |
| $M$                                                                  | Sequence length after tokenization          |
| $d$                                                                  | Token embedding dimension                   |
| $\mathcal{Y}_{train}$                                                | Training nodes                              |
| $\mathcal{Y}_{test}$                                                 | Test nodes                                  |
| $C$                                                                  | Number of classes                           |
| $\mathcal{L}$                                                        | Loss function used to train the framework   |
| $\mathcal{L}_{ce}$                                                   | Cross-entropy loss                          |
| $\mathcal{L}_{mse}$                                                  | Mean-squared error loss                     |
| $\mathbf{X}_{lin} \in \mathbb{R}^{M \times d_x}$                     | Linear layer input before quantization      |
| $\tilde{\mathbf{X}}_{lin} \in \mathbb{R}^{M \times d_x}$             | Linear layer input after quantization       |
| $\mathbf{Y}_{lin} \in \mathbb{R}^{M \times d_y}$                     | Linear layer output before quantization     |
| $\tilde{\mathbf{Y}}_{lin} \in \mathbb{R}^{M \times d_y}$             | Linear layer output after quantization      |
| $\mathbf{W}_{lin} \in \mathbb{R}^{d_x \times d_y}$                   | Linear layer parameters before quantization |
| $\tilde{\mathbf{W}}_{lin} \in \mathbb{R}^{d_x \times d_y}$           | Linear layer parameters after quantization  |
| $d_x$                                                                | Hidden dimension of linear layer input      |
| $d_y$                                                                | Hidden dimensions of linear layer output    |
| $\tilde{\mathbf{Z}}_{\text{BitTrans}} \in \mathbb{R}^{M \times d_y}$ | Quantized BitTransformer output             |
| $\hat{\mathbf{Z}}_{\text{BitTrans}} \in \mathbb{R}^{M \times d_y}$   | Dequantized BitTransformer output           |
| $\text{Quant}(\cdot)$                                                | Element-wise quantization function          |
| $\text{Dequant}(\cdot)$                                              | Element-wise dequantization function        |
| $\ \mathbf{X}_{lin}\ _{\infty}$                                      | $l_{\infty}$ norm of $\mathbf{X}_{lin}$     |
| $\mathbf{X} \in \mathbb{R}^{N \times M \times d}$                    | Sequence input after embedding layer        |
| $\mathbf{Y} \in \mathbb{R}^{N \times C}$                             | Target classes                              |
| $\mathbf{U}_{seq} \in \mathbb{R}^{N \times d}$                       | Sequence embeddings                         |
| $\tilde{\mathbf{W}}_{\text{Trans}}$                                  | Set of quantized transformer weights        |
| $\mathbf{W}_{\text{GNN}}$                                            | Set of GNN weights                          |

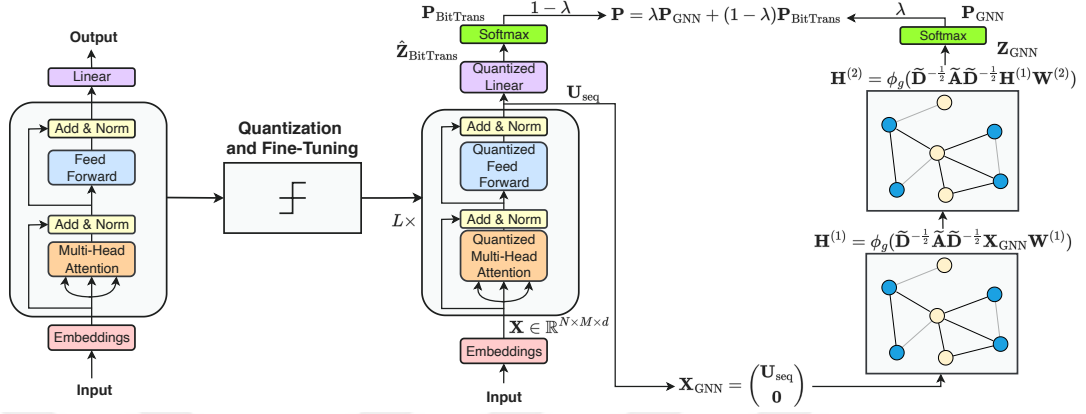


Figure 3.1: Proposed BitTransGNN architecture, including transformer quantization and fine-tuning, output production of BitTransformer and GNN modules, and the integration of the two models. The schematic contains a BitTransGNN model with an  $L$ -layered BitTransformer and a 2-layered GNN that employs graph convolution.

## 3.1 BitTransformer Module

We first outline the procedure that we use to quantize the transformer model. Then, we formulate the information processing conducted across quantized transformer blocks within the BitTransGNN framework. Lastly, we explain the integration of quantization to the fine-tuning stage of the transformer model.

### 3.1.1 Quantization Procedure

We quantize the real-valued weights of the transformer network to low-bit precision following the approaches in [98]. We map each value in the weight matrices to an element in the quantization set  $\mathcal{S}$ . Elements of  $\mathcal{S}$  are determined based on the quantization rate. The bit-precision of quantization and the corresponding sets are displayed in Table 3.2.

The conventional matrix multiplication in a linear transformation layer is performed as following:

$$\mathbf{Y}_{\text{lin}} = \mathbf{X}_{\text{lin}} \mathbf{W}_{\text{lin}}, \quad (3.1)$$

Table 3.2: Sets used to quantize the full-precision weights.

|               | <b>1-bit</b> | <b>1.58-bit</b> | <b>2-bit</b>       | <b>2.32-bit</b>       |
|---------------|--------------|-----------------|--------------------|-----------------------|
| $\mathcal{S}$ | $\{-1, 1\}$  | $\{-1, 0, 1\}$  | $\{-2, -1, 1, 2\}$ | $\{-2, -1, 0, 1, 2\}$ |

where  $\mathbf{X}_{\text{lin}} \in \mathbb{R}^{M \times d_x}$ ,  $\mathbf{Y}_{\text{lin}} \in \mathbb{R}^{M \times d_y}$  and  $\mathbf{W}_{\text{lin}} \in \mathbb{R}^{d_x \times d_y}$  are the input, output and weight matrices in the linear layers. We apply element-wise quantization to the weight matrix as:

$$\widetilde{\mathbf{W}}_{\text{lin}} = \text{Clamp} \left( \text{round} \left( \frac{\mathbf{W}_{\text{lin}}}{\alpha_{\text{W}}} \right), \min(\mathcal{S}), \max(\mathcal{S}) \right), \quad (3.2)$$

$$\text{Clamp}(x, a, c) = \min(\max(x, a), c), \quad x, a, c \in \mathbb{R}. \quad (3.3)$$

$\text{round}(x) : \mathbb{R} \rightarrow \mathbb{Z}$  maps the real-valued number  $x$  to the nearest integer. We quantize the weights using absmean quantization. In absmean quantization, the scaling factor  $\alpha_{\text{W}}$  is defined as following:

$$\alpha_{\text{W}} = \frac{\mu}{\max(\mathcal{S})}, \quad (3.4)$$

$$\mu = \frac{1}{nm} \sum_i \sum_j |(\mathbf{W}_{\text{lin}})_{ij}|. \quad (3.5)$$

In addition to weight quantization, we also quantize the activations within the transformer model to  $b = 8$ -bits by mapping them into the set  $\mathcal{S}_{\text{X}} = \{x \in \mathbb{Z} \mid -2^{b-1} \leq x < 2^{b-1}\}$ :

$$\widetilde{\mathbf{X}}_{\text{lin}} = \text{Clamp} \left( \text{round} \left( \frac{\mathbf{X}_{\text{lin}}}{\alpha_{\text{X}}} \right), \min(\mathcal{S}_{\text{X}}), \max(\mathcal{S}_{\text{X}}) \right). \quad (3.6)$$

Different from weight quantization, we use absmax quantization over the activations, and the scaling factor  $\alpha_{\text{X}}$  is defined as following for this approach:

$$\alpha_{\text{X}} = \frac{\|\mathbf{X}_{\text{lin}}\|_{\infty}}{\max(\mathcal{S}_{\text{X}})}. \quad (3.7)$$

$\|\mathbf{X}_{\text{lin}}\|_{\infty}$  refers to the  $l_{\infty}$  norm of  $\mathbf{X}_{\text{lin}}$ . Through quantization, the linear layer in Eq. (3.1) is converted to the following:

$$\widetilde{\mathbf{Y}}_{\text{lin}} = \text{Quant}(\mathbf{Y}_{\text{lin}}) = \widetilde{\mathbf{X}}_{\text{lin}} \widetilde{\mathbf{W}}_{\text{lin}}. \quad (3.8)$$

Finally, after the quantized matrix multiplication, the output of the quantized linear layer is dequantized back to its original precision before being provided to the later layers of the model:

$$\hat{\mathbf{Y}}_{\text{lin}} = \text{Dequant}(\tilde{\mathbf{Y}}_{\text{lin}}) = \alpha_W \alpha_X \times \tilde{\mathbf{Y}}_{\text{lin}}. \quad (3.9)$$

$\text{Quant}(\cdot)$  and  $\text{Dequant}(\cdot)$  will be used to refer to the quantization and dequantization functions in Eqs. (3.8) and (3.9), respectively. We use the straight-through estimator (STE) [99] to approximate the gradient of binary parameters, as the quantization function in Eq. (3.3) is non-differentiable and incompatible with backpropagation.

We quantize all linear transformations within transformer layers, including the feed-forward, attention, and pooling layers, as well as the ones in the linear classifier that is linked to the transformer model. Since its look-up table nature is vastly different from matrix multiplications, embeddings were found to be more sensitive to quantization. Following similar studies [32], we maintain the embeddings full-precision; however, Eq. (3.2) can also be used to quantize the embedding weights and further reduce the parameter overhead of the transformer model.

### 3.1.2 Quantized Transformer Blocks

The quantized transformer first receives  $N$  raw sequences. The model tokenizes each sequence by mapping sequence elements to tokens. In addition to mapping the data entities, tokenization also adds the special tokens [CLS], [SEP], and [PAD] to the sequence. The [CLS] token, placed at the beginning of the sequence, is used to aggregate information from other sequence elements. The [SEP] token is used to mark the boundary between different parts of the sequence. Lastly, the [PAD] token is appended to the end of each sequence as a padding to standardize the token count across sequences. After tokenization, each sequence has a uniform length of  $M$ . The model then maps the tokens to an embedding matrix  $\mathbf{X} \in \mathbb{R}^{N \times M \times d}$  through its embedding layer, where  $d$  is the embedding dimension. The

embedding matrix is supplied to the quantized transformer model as input:

$$\mathbf{H}^{(0)} = \mathbf{X}. \quad (3.10)$$

The quantized transformer contains  $L$  blocks containing multi-head self-attention (MHSA) mechanisms and feed-forward networks (FFNs). The output of each quantized transformer block (QTB) is passed to the next block as input:

$$\mathbf{H}^{(l)} = \text{QTB}(\mathbf{H}^{(l-1)}) \quad \forall l \in \{1, 2, \dots, L\}. \quad (3.11)$$

The computations within each QTB are uniform. We display the computations for an arbitrary  $l$ th block as an illustrative example.

The block first processes  $\mathbf{H}^{(l-1)}$  through the MHSA mechanism [2]. MHSA processes the embedding matrix  $\mathbf{H}^{(l-1)}$  in  $h$  independent attention heads. The quantized transformer projects the embeddings matrix to three different matrices within each head: the query  $\tilde{\mathbf{Q}}_i^{(l)}$ , key  $\tilde{\mathbf{K}}_i^{(l)}$  and value  $\tilde{\mathbf{V}}_i^{(l)}$ . Projections are applied through quantized linear transformations:

$$\begin{aligned} \tilde{\mathbf{Q}}_i^{(l)} &= \tilde{\mathbf{H}}_i^{(l-1)} \tilde{\mathbf{W}}_{Q_i}^{(l)}, \quad \tilde{\mathbf{K}}_i^{(l)} = \tilde{\mathbf{H}}_i^{(l-1)} \tilde{\mathbf{W}}_{K_i}^{(l)}, \\ \tilde{\mathbf{V}}_i^{(l)} &= \tilde{\mathbf{H}}_i^{(l-1)} \tilde{\mathbf{W}}_{V_i}^{(l)}. \end{aligned} \quad (3.12)$$

$\tilde{\mathbf{W}}_{Q_i}^{(l)}, \tilde{\mathbf{W}}_{K_i}^{(l)}, \tilde{\mathbf{W}}_{V_i}^{(l)} \in \mathbb{R}^{d \times \frac{d}{h}}$  are learnable quantized weights. MHSA computes attention scores at each attention head using the three matrices:

$$\text{Att}_i \left( \hat{\mathbf{Q}}_i^{(l)}, \hat{\mathbf{K}}_i^{(l)}, \hat{\mathbf{V}}_i^{(l)} \right) = \text{SOFTMAX} \left( \frac{\hat{\mathbf{Q}}_i^{(l)} \hat{\mathbf{K}}_i^{(l)T}}{\sqrt{d/h}} \right) \hat{\mathbf{V}}_i^{(l)}, \quad (3.13)$$

where  $\hat{\mathbf{Q}}_i^{(l)}, \hat{\mathbf{K}}_i^{(l)}, \hat{\mathbf{V}}_i^{(l)}$  are the dequantized versions of  $\tilde{\mathbf{Q}}_i^{(l)}, \tilde{\mathbf{K}}_i^{(l)}, \tilde{\mathbf{V}}_i^{(l)}$ , and the multiplications between the three matrices are maintained full-precision. The attention scores calculated at each head are then concatenated and passed through a quantized linear transformation:

$$\widetilde{\text{MHA}} \left( \hat{\mathbf{Q}}^{(l)}, \hat{\mathbf{K}}^{(l)}, \hat{\mathbf{V}}^{(l)} \right) = \text{Quant} \left( \left[ \text{head}_1^{(l)} \dots \text{head}_h^{(l)} \right] \right) \tilde{\mathbf{W}}_o^{(l)}, \quad (3.14)$$

where  $\text{head}_i^{(l)} = \text{Att}_i(\hat{\mathbf{Q}}_i^{(l)}, \hat{\mathbf{K}}_i^{(l)}, \hat{\mathbf{V}}_i^{(l)})$  and  $\tilde{\mathbf{W}}_o^{(l)} \in \mathbb{R}^{d \times d}$  is a learnable quantized weight. The MHSA output is then passed through a layer normalization (LN) component with residual connection:

$$\mathbf{H}_{\text{att}}^{(l)} = \text{LN} \left( \mathbf{H}^{(l-1)} + \text{Dequant} \left( \widetilde{\text{MHA}} \left( \hat{\mathbf{Q}}^{(l)}, \hat{\mathbf{K}}^{(l)}, \hat{\mathbf{V}}^{(l)} \right) \right) \right). \quad (3.15)$$

Since they have negligible costs, layer normalizations and residual connections are maintained full-precision. The output of the layer normalization is then supplied to a quantized FFN:

$$\tilde{\mathbf{H}}_{\text{FNN}}^{(l)} = \phi_T \left( \text{Quant} \left( \mathbf{H}_{\text{att}}^{(l)} \right) \widetilde{\mathbf{W}}_{\text{FFN}}^{(l)} \right), \quad (3.16)$$

where  $\phi_T(\cdot) = \text{GELU}(\cdot)$  is the nonlinear activation used within the QTBs and  $\widetilde{\mathbf{W}}_{\text{FFN}}^{(l)} \in \mathbb{R}^{d \times d}$  are the learnable quantized weights of the FFN. The FFN output is again passed through a layer normalization component with residual connection to obtain the final output of the  $l$ th QTB:

$$\mathbf{H}^{(l)} = \text{LN} \left( \mathbf{H}_{\text{att}}^{(l)} + \text{Dequant} \left( \tilde{\mathbf{H}}_{\text{FNN}}^{(l)} \right) \right). \quad (3.17)$$

Following Eq. (3.11), the output of the  $L$ th QTB  $\mathbf{H}^{(L)} \in \mathbb{R}^{N \times M \times d}$  is obtained. During tokenization, the [CLS] token, which served to aggregate information from other sequence elements, was put to the first entry of each sequence. The BitTransformer produces an embedding for the [CLS] token at its output, which we extract from  $\mathbf{H}_{:,0,:}^{(L)}$  to construct sequence embeddings  $\mathbf{U}_{\text{seq}} \in \mathbb{R}^{N \times d}$ .  $\mathbf{U}_{\text{seq}}$  is finally passed through a quantized linear classifier to obtain the BitTransformer transformer output:

$$\hat{\mathbf{Z}}_{\text{BitTrans}} = \text{Dequant} \left( \tilde{\mathbf{U}}_{\text{seq}} \widetilde{\mathbf{W}}_{\text{cls}} \right), \quad (3.18)$$

where  $\tilde{\mathbf{U}}_{\text{seq}} = \text{Quant}(\mathbf{U}_{\text{seq}})$  and  $\widetilde{\mathbf{W}}_{\text{cls}} \in \mathbb{R}^{d \times C}$  is the quantized linear classifier weights.  $\hat{\mathbf{Z}}_{\text{BitTrans}} \in \mathbb{R}^{N \times 1}$  is used as the final output in regression tasks. In classification tasks, the output embeddings are additionally passed through a **SOFTMAX** function to obtain probabilities over class distributions:

$$\mathbf{P}_{\text{BitTrans}} = \text{SOFTMAX} \left( \hat{\mathbf{Z}}_{\text{BitTrans}} \right). \quad (3.19)$$

### 3.1.3 Integrating Quantization and Fine-Tuning

We fine-tune and quantize the transformer model before incorporating pre-trained transformer models into BitTransGNN. We investigate the integration of quantization to transformer fine-tuning in two different stages.

1. **Quantization Aware Fine-Tuning (QAFT)** quantizes pre-trained model weights to low-bit precision and then fine-tunes the transformer after quantization.
2. **Post Fine-Tuning Quantization (PFTQ)** conducts fine-tuning using full-precision parameters and quantizes transformer parameters after the fine-tuning is completed.

QAFT optimizes the model over the quantized set of weights, while PFTQ performs training using full-precision parameters and then quantizes model weights. PFTQ is simpler to apply than QAFT because it does not require additional training with the quantized weights; however, since the model does not optimize over the quantized set of weights, it is generally not feasible to use the model directly with low-bit PFTQ, as it significantly lowers performance. We compare the use of QAFT and PFTQ on pre-trained BERT and RoBERTa transformer models in Table 3.3. We indicate the bit-precision of quantization alongside the model label in parentheses. When binarized after fine-tuning, transformers generally perform slightly better than random guessing, which is particularly apparent for 20NG, MR, and Ohsumed datasets. For higher bit-precisions, the model starts to produce more sensible results with PFTQ, as evident in the performance of 2.32-bit quantized variants; however, the performance is still significantly worse than QAFT in all datasets.

## 3.2 GNN Module

We first formulate the construction of the text-graph from a given corpus. Then, we explain how GNN blocks process text data through the text-graph structure to produce an output.

Table 3.3: Performance of quantized models when Quantization-Aware Fine-Tuning (QAFT) and Post Fine-Tuning Quantization (PFTQ) are used as fine-tuning strategies.

| BitTransformer     | 20NG  | MR    | R8    | R52   | Ohsumed |
|--------------------|-------|-------|-------|-------|---------|
| <b>QAFT</b>        |       |       |       |       |         |
| BERT (1-bit)       | 79.51 | 76.92 | 97.30 | 94.42 | 63.34   |
| BERT (1.58-bit)    | 80.62 | 77.66 | 97.29 | 95.43 | 64.08   |
| BERT (2.32-bit)    | 82.37 | 81.86 | 96.82 | 94.85 | 66.62   |
| RoBERTa (1-bit)    | 81.49 | 77.27 | 97.27 | 94.77 | 61.99   |
| RoBERTa (1.58-bit) | 82.14 | 79.49 | 97.48 | 95.62 | 64.38   |
| RoBERTa (2.32-bit) | 83.05 | 83.28 | 97.60 | 95.61 | 66.08   |
| <b>PFTQ</b>        |       |       |       |       |         |
| BERT (1-bit)       | 10.49 | 50.73 | 77.80 | 67.91 | 15.38   |
| BERT (1.58-bit)    | 14.05 | 60.38 | 79.76 | 69.43 | 28.99   |
| BERT (2.32-bit)    | 64.83 | 80.14 | 89.58 | 76.91 | 23.72   |
| RoBERTa (1-bit)    | 5.32  | 50.00 | 3.15  | 0.43  | 5.66    |
| RoBERTa (1.58-bit) | 7.62  | 50.17 | 4.25  | 26.91 | 7.74    |
| RoBERTa (2.32-bit) | 31.53 | 61.06 | 79.67 | 74.80 | 21.99   |

### 3.2.1 Graph Generation

To create a text graph from a corpus, we place each word and document in the corpus on the nodes of a heterogeneous graph. We construct the edges between different nodes using text statistics. We use positive point-wise mutual information (PPMI) and term frequency-inverse document frequency (TF-IDF) metrics to determine word-word and word-document relations, respectively. The adjacency matrix of the text-graph we use can be formulated as following [83]:

$$\mathbf{A}(i, j) = \begin{cases} \text{PPMI}(i, j), & \text{if } i, j \text{ are words, } \text{PPMI}(i, j) > 0 \\ \text{TF-IDF}(i, j), & \text{if } i \text{ is document, } j \text{ is word} \\ 1, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases} \quad (3.20)$$

### 3.2.2 GNN Blocks

The GNN module is constructed by stacking multiple message-passing layers that operate on graph-structured data. The feature update equation of a generic message-passing layer can be formulated as following:

$$\mathbf{F}^{(l)} = \phi_G \left( \mathbf{S}^{(l)} \mathbf{F}^{(l-1)} \mathbf{W}^{(l)} \right), \quad (3.21)$$

where  $l$  is the layer number,  $\mathbf{S}^{(l)}$  is a graph structure encoding,  $\mathbf{F}^{(l-1)}$  denotes the hidden layer features,  $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$  is the trainable weight matrix,  $d^{(l)}$  corresponds to the hidden feature dimension at layer  $l$ , and  $\phi_G(\cdot)$  is the nonlinear activation function used within GNN layers.

To structure our message-passing components, we use graph convolutional layers. For a graph  $G = (\mathcal{V}, \mathcal{E})$ , where  $V$  is the set of nodes and  $\mathcal{E}$  is the set of edges, the graph adjacency matrix  $\mathbf{A} \in \mathbb{R}^{N \times N}$  defines the connectivity across  $N$  graph nodes. In a graph convolutional layer, the graph structure encoding is formulated based on the graph adjacency matrix, as  $\mathbf{S}^{(l)} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ , where  $\mathbf{D}(i, i) = \sum_j \mathbf{A}(i, j)$  is the degree matrix of  $\mathbf{A}$ . As a result, we use the following update equation in the  $l$ th layer of our GNN module:

$$\mathbf{F}^{(l)} = \phi_G \left( \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{F}^{(l-1)} \mathbf{W}^{(l)} \right). \quad (3.22)$$

We use  $\phi_G(\cdot) = \text{ReLU}(\cdot)$  activation across GNN layers.

The sequence embeddings  $\mathbf{U}_{\text{seq}}$  outputted by the quantized transformer are supplied to the graph module as inputs. The embeddings of the sequence nodes are initialized using  $\mathbf{U}_{\text{seq}} \in \mathbb{R}^{N \times d}$ , and the node embeddings of the  $T$  token types within the corpora are initialized as  $\mathbf{0} \in \mathbb{R}^{T \times d}$ :

$$\mathbf{F}^{(0)} = \begin{pmatrix} \mathbf{U}_{\text{seq}} \\ \mathbf{0} \end{pmatrix} \in \mathbb{R}^{(N+T) \times d}. \quad (3.23)$$

The embeddings are processed by the GNN module containing  $K$  stacked graph convolutional layers defined in Eq. (3.22):

$$\mathbf{F}^{(K)} = \text{GNN}(\mathbf{U}_{\text{seq}}). \quad (3.24)$$

Finally, the embeddings of the  $N$  document nodes are extracted from  $\mathbf{F}^{(K)} \in \mathbb{R}^{(N+T) \times C}$  to obtain  $\mathbf{Z}_{\text{GNN}} \in \mathbb{R}^{N \times C}$ .  $\mathbf{Z}_{\text{GNN}}$  is used as the final GNN output in regression tasks. In classification tasks, the embeddings  $\mathbf{Z}_{\text{GNN}}$  are additionally passed through a **SOFTMAX** function:

$$\mathbf{P}_{\text{GNN}} = \text{SOFTMAX}(\mathbf{Z}_{\text{GNN}}). \quad (3.25)$$

### 3.3 The Integration Module

Despite holding a significant potential to reduce the memory and energy costs of transformers, quantization leads to a notable degradation in the representational capabilities of transformers. In addition, although successfully representing the local context, transformers are poor at capturing global relationships across data entities that are beyond the scope of their context windows. To reduce the shortcomings of quantized transformers, BitTransGNN employs a GNN module in conjunction with the BitTransformer, as can be seen visually in Fig. 3.1. GNN uses the sequence embeddings supplied by the BitTransformer and operates over these embeddings to enhance BitTransformer outputs. The integration module synthesizes the outputs of the BitTransformer and the GNN models to improve BitTransformer training.

In classification tasks, BitTransGNN integrates the predictions produced by the BitTransformer and the GNN as following:

$$\mathbf{P}_{\text{BitTransGNN}} = \lambda \mathbf{P}_{\text{GNN}} + (1 - \lambda) \mathbf{P}_{\text{BitTrans}}. \quad (3.26)$$

$\lambda \in [0, 1]$  is an hyperparameter governing the contribution of the two models. In classification tasks, we use the cross-entropy loss to account for the training loss  $\mathcal{L}$ :

$$\mathcal{L}_{ce} = -\frac{1}{|\mathcal{Y}_{train}|} \sum_{d \in \mathcal{Y}_{train}} \sum_{c=1}^C \mathbf{Y}_{dc} \ln(\mathbf{P}_{\text{BitTransGNN}})_{dc}, \quad (3.27)$$

where  $(\mathbf{P}_{\text{BitTransGNN}})_{dc}$  is the probability predicted for class  $c$  of document  $d$ ,  $\mathbf{Y}_{dc}$  is the ground-truth label of the corresponding instance,  $C$  is the number of classes, and  $\mathcal{Y}_{train}$  is the set of labeled document indices.

In regression tasks, the embeddings  $\mathbf{Z}_{\text{GNN}}$  and  $\hat{\mathbf{Z}}_{\text{BitTrans}}$  before the **SOFTMAX** function are used to obtain the final output:

$$\mathbf{Z}_{\text{BitTransGNN}} = \lambda \mathbf{Z}_{\text{GNN}} + (1 - \lambda) \hat{\mathbf{Z}}_{\text{BitTrans}}, \quad (3.28)$$

and the model is trained using the mean squared error (MSE):

$$\mathcal{L}_{mse} = \frac{1}{2|\mathcal{Y}_{train}|} \sum_{d \in \mathcal{Y}_{train}} (\mathbf{Y}_d - \mathbf{Z}_d)^2. \quad (3.29)$$

## 3.4 Training Schemes

We propose two training schemes,  $\text{BitTransGNN}_{\text{Static}}$  and  $\text{BitTransGNN}_{\text{Dynamic}}$ , to train the BitTransGNN framework.

### 3.4.1 $\text{BitTransGNN}_{\text{Static}}$

The static method considers the pre-trained quantized transformer module in a fixed state and conducts training only over the GNN model. The quantized transformer initializes the embeddings of the document nodes and the GNN module operates over these node embeddings. GNN only serves to improve the final outputs without an influence over the quantized transformer parameters, since the quantized transformer is not trained in this method.  $\text{BitTransGNN}_{\text{Static}}$  optimizes GNN parameters on the following objective:

$$\arg \min_{\mathbf{W}_{\text{GNN}}} \mathcal{L}$$

where  $\mathcal{L}$  refers to the training loss. We only integrate QAFT BitTransformer models to the static method since  $\text{BitTransGNN}_{\text{Static}}$  does not involve any BitTransformer training.

### 3.4.2 BitTransGNN<sub>Dynamic</sub>

The dynamic training scheme jointly trains the quantized transformer model and the GNN. There is a major difference in the training procedures of GNN and transformer models. While GNN is trained with full-batch gradient descent, transformer models cannot be trained in that manner due to memory constraints, which imposes a difficulty in the joint training of the two models. To address this issue, we modify the training procedure to be able to train the two models jointly. At each iteration, a mini-batch of document nodes are sampled from the text graph. The embeddings of these nodes are updated using the document embeddings  $\mathbf{U}_{\text{seq}}$  outputted by the quantized transformer at that instance, and the GNN operates over the updated node features. The document embeddings are also updated at the beginning of each epoch based on the state of the quantized transformer at that instant. Dynamic training enables one to investigate whether the quantized model can find better parameters when trained jointly with a GNN. BitTransGNN<sub>Dynamic</sub> optimizes the quantized transformer and GNN parameters jointly on the following training objective:

$$\arg \min_{(\mathbf{w}_{\text{GNN}}, \tilde{\mathbf{w}}_{\text{Trans}})} \mathcal{L}$$

We explore the use of both QAFT and PFTQ BitTransformer models in the dynamic training scheme.

## 3.5 Inductive BitTransGNN Variants

In this sub-chapter, we explore potential improvements regarding the utility of the BitTransGNN framework and present methods to remove the possible shortcomings of our model. Due to the transductive essence of the GNN model used as part of the framework, BitTransGNN is inherently transductive. Despite the advantages it provides, this characteristic also poses some limitations. Because it is transductive, the joint BitTransGNN model cannot generalize to unseen node instances and has to be re-trained in case new data entities are presented.

This limits the functionality of the method and contradicts the inductive nature of transformer models. Additionally, despite the improvements it provides, BitTransGNN makes use of the GNN model weights on top of the quantized transformer parameters while producing its output. This creates a difficulty in assessing whether the framework actually enhances quantized transformer training or whether the improvements are only a natural consequence of the increased parameter count due to the additional GNN parameters. To generalize the use of our model to inductive problem settings, we propose two methods that encapsulate the knowledge learned by a BitTransGNN model during training within a quantized transformer model. Both methods allow to transfer the knowledge learned by the BitTransGNN framework into a quantized transformer that can operate solitarily in inductive problem settings with zero additional inference costs.

The first method directly detaches the quantized transformer and its linear classifier from BitTransGNN after training and uses them for inference. While this extension allows the quantized transformer to infer on unseen data with zero additional training and inference costs, since the original optimization objective is defined over the joint architecture, the transformer might perform poorly when directly separated from the GNN. To enhance the separation, we design a knowledge distillation scheme that compresses the information learned by BitTransGNN into a solitary quantized transformer. Through the two methods, the quantized transformer can infer inductively while leveraging the knowledge captured by the joint BitTransGNN model with zero additional costs in inference time. Both methods allow to use the quantized transformer solitarily without any influence from the GNN during inference time.

### **3.5.1 Direct Separation (DS)**

In Direct Separation (DS), we separate the quantized transformer and its linear classifier from BitTransGNN after training and use them directly for inference. DS uses these components directly for inference without any additional training.

After DS, BitTransGNN output takes the following form:

$$\hat{\mathbf{Z}}_{\text{BitTrans}} = \text{Dequant} \left( \tilde{\mathbf{U}}_{\text{seq}} \tilde{\mathbf{W}}_{\text{cls}} \right). \quad (3.30)$$

$$\mathbf{P} = \mathbf{P}_{\text{BitTrans}} = \text{SOFTMAX} \left( \hat{\mathbf{Z}}_{\text{BitTrans}} \right). \quad (3.31)$$

We employ DS only over BitTransGNN<sub>Dynamic</sub> methods, as BitTransGNN<sub>Static</sub> does not train the BitTransformer model and would return the same output as the scenario where the BitTransformer model is trained independently, without using the BitTransGNN framework.

### 3.5.2 Induction Through Knowledge Distillation (KD)

Taking into account the potential tradeoffs regarding the direct separation of the quantized transformer from the joint architecture, we additionally introduce a knowledge distillation framework to enhance the performance of the solitary quantized transformer. To this end, we employ a teacher-student framework. We utilize a trained BitTransGNN variant as the teacher and a solitary BitTransformer as the student. We apply offline distillation [45, 100] to limit the method’s overhead and only train the student quantized transformer. The intermediate layers of GNN and transformer are vastly different. Hence, we only build correspondence between the model outputs and use a response-based distillation scheme [45, 101–103]. We present a visual summary of the formulation of the distillation loss in Fig. 3.2.

The student model is trained on both the distillation loss and the ground-truth loss. We compute the distillation loss  $\mathcal{L}_d$  over the logits of the student and teacher models, and the ground-truth loss  $\mathcal{L}_{\text{truth}}$  by comparing the student model outputs against ground-truth labels. The combined loss function  $\mathcal{L}$  used to train the student model is formulated as following:

$$\mathcal{L} = \alpha_d \mathcal{L}_d + (1 - \alpha_d) \mathcal{L}_{\text{truth}}. \quad (3.32)$$

$\alpha_d \in [0, 1]$  is an hyperparameter that determines the influence of the distillation loss and the ground-truth loss on the joint loss function. Using larger values of  $\alpha_d$

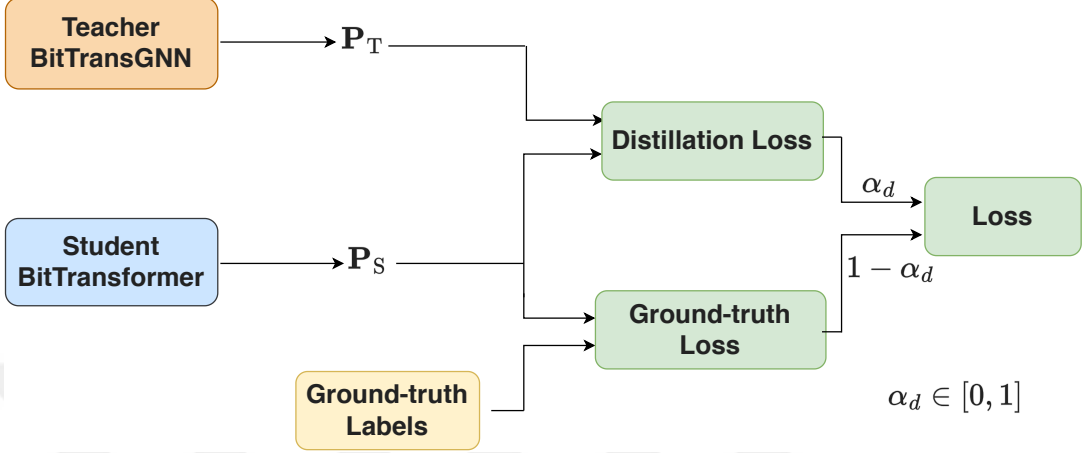


Figure 3.2: The computation of the knowledge distillation loss.

increases the influence of the distillation loss, while smaller values of  $\alpha_d$  increase the contribution of the ground-truth loss on student model training.

In classification tasks, we use the Kullback-Leibler (KL) divergence metric to account for the distillation loss. KL divergence measures the difference between two probability distributions. We want the student model to mimic the teacher model. To achieve this goal, we want to minimize the KL divergence between the output distributions of the two models during training. For the output probability distributions of the student model  $\mathbf{P}_S$  and the teacher  $\mathbf{P}_T$ , the KL divergence is computed over  $C$  different classes as follows:

$$\text{KL}(\mathbf{P}_T, \mathbf{P}_S) = \sum_{n=1}^N \sum_{c=1}^C (\mathbf{P}_T)_{nc} \log \frac{(\mathbf{P}_S)_{nc}}{(\mathbf{P}_T)_{nc}}. \quad (3.33)$$

The standard cross-entropy loss converts model outputs into probabilities using the softmax function. In KD, we use a modified softmax function with an additional temperature hyperparameter  $\tau$  to convert model outputs into probabilities.  $\tau$  determines how spread the output probability distribution will be. Using a  $\tau \geq 1$  softens the outputs distributions of the two models to provide more information during training. On the contrary, smaller values of  $\tau$  produce distributions that are largely concentrated over a single class. Softmax with temperature is defined as following for some matrix  $\mathbf{Z}$ :

$$\text{SOFTMAX}_{\tau}(\mathbf{Z}) = \text{SOFTMAX} \left( \frac{\mathbf{Z}}{\tau} \right). \quad (3.34)$$

Using this definition, the probability  $\mathbf{P}_{nc}$  for class  $c$  of sequence  $n$  is computed as follows:

$$\mathbf{P}_{nc} = \text{SOFTMAX}_\tau(\mathbf{Z}_{nc}) = \sum_{n=1}^N \sum_{c=1}^C \frac{\exp((\mathbf{Z})_{nc}/\tau)}{\sum_{k=1}^C \exp((\mathbf{Z})_{nk}/\tau)}. \quad (3.35)$$

The distillation loss between the softened output distributions of the teacher BitTransGNN and the student BitTransformer is computed as follows:

$$\mathcal{L}_d = \frac{\tau^2}{N} \text{KL}(\mathbf{P}_T, \mathbf{P}_S), \quad (3.36)$$

where  $\mathbf{P}_T = \text{SOFTMAX}_\tau(\mathbf{Z}_T)$  and  $\mathbf{P}_S = \text{SOFTMAX}_\tau(\mathbf{Z}_S)$  for some temperature  $\tau$ . We scale the distillation loss by the factor  $\tau^2$  to compensate for the scaling effect of the temperature on the gradients during backpropagation [45].

In regressions tasks, we formulate the distillation loss as the MSE loss between the teacher and the student model outputs:

$$\mathcal{L}_d = \frac{1}{2N} \sum_{n=1}^N ((\mathbf{Z}_T)_n - (\mathbf{Z}_S)_n)^2. \quad (3.37)$$

After the training is completed, only the outputs of the student BitTransformer model  $\mathbf{P}_S$  and  $\mathbf{Z}_S$  are used for inference.

# Chapter 4

## Efficiency Analysis

In this chapter, we perform a theoretical analysis to display the potential efficiency advantages provided by our architecture in terms of parameter memory overhead and energy consumption.

### 4.1 Memory Efficiency

Full-precision transformer parameters are generally stored in 32 bits. Quantization can reduce the memory required to store model parameters by up to 32 times. To verify this potential, we calculate the memory expense of hosting Bit-Transformer and BitTransGNN variants.

We present a list of architectural hyperparameters contained within BERT and RoBERTa transformer variants in Table 4.1 that we use to calculate the memory efficiency of each model. We also display the total number of parameters in each BERT and RoBERTa component, along with the number of parameters within a graph convolutional layer in Table 4.2. Other than the variables listed in Table 4.1,  $d_g$  refers to the hidden layer dimension of the GNN model and  $d_o$  refers to the number of output neurons, i.e., number of output classes.

Table 4.1: Exact sizes of BERT and RoBERTa transformer components.

|                                      | BERT  | RoBERTa |
|--------------------------------------|-------|---------|
| Vocabulary Size ( $V$ )              | 30522 | 50265   |
| Maximum Sequence Length ( $S$ )      | 512   | 514     |
| Token Type Embeddings ( $T$ )        | 2     | 1       |
| Hidden Dimension ( $d_h$ )           | 768   | 768     |
| Feed-Forward Dimension ( $d_f$ )     | 3072  | 3072    |
| Attention Hidden Dimension ( $d_s$ ) | 64    | 64      |
| Number of Attention Heads ( $H$ )    | 12    | 12      |
| Number of Attention Layers ( $L$ )   | 12    | 12      |

Table 4.2: Number of parameters and the number of bits allocated for parameters contained within different BERT transformer components and the convolutional GNN model. We generally use the 20NG model as reference, which has  $d_o = 20$ .

|                      | # of Parameters                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------|
| Embeddings           | $p_e = (V \times d_h) + (S \times d_h) + (T \times d_h) + (d_h + d_h)$                             |
| Attention            | $p_a = L \times (H \times 3 \times (d_h \times d_s + d_s) + (d_h \times d_h + d_h) + (d_h + d_h))$ |
| Feed Forward Network | $p_f = L \times ((d_h \times d_f + d_f) + (d_f \times d_h + d_h) + (d_h + d_h))$                   |
| Pooler               | $p_p = d_h \times d_h + d_h$                                                                       |
| Classifier           | $p_c = d_h \times d_o + d_o$                                                                       |
| GNN Layer            | $p_g = (d_h \times d_g + d_g) + (d_g \times d_o + d_o)$                                            |

The number of bits required to store the parameters of a quantized transformer model can be calculated as follows:

$$B_{\text{BitTrans}} = p_e \times b_e + p_a \times b_l + p_f \times b_l + p_p \times b_l + p_c \times b_l, \quad (4.1)$$

where  $B_{\text{BitTrans}}$  refers to the number of bits required to contain BitTransformer parameters and  $p_e, p_a, p_f, p_p, p_c$  represent the number of parameters in the embedding, attention, feed-forward, pooling, and classifier layers, respectively.  $b_e$  refers to the number of bits used to represent embedding module weights and  $b_l$  is the number of bits used to store the linear layer weights. We use a full-precision embedding module and quantized linear layer weights such that for a 1-bit BitTransformer,  $b_l = 1$  and  $b_e = 32$ . We calculate the number of bits required to

store BitTransGNN parameters as following:

$$B_{\text{BitTransGNN}} = B_{\text{BitTrans}} + p_g \times b_g. \quad (4.2)$$

$B_{\text{BitTransGNN}}$  refers to the number of bits used to store BitTransGNN parameters,  $p_g$  refers to the number of parameters within the GNN module and  $b_g$  refers to the bit-precision used to store GNN parameters. We use a full-precision GNN model in our experiments, i.e.,  $b_g = 32$ .

The memory required to store the parameters of BERT-based baseline models and BitTransGNN variants is listed in Table 4.3 for different quantization scenarios.  $W\{b_l\}E\{b_e\}$  refers to an instance where the linear layers are quantized to  $b_l$  bits and the embedding module is quantized to  $b_e$  bits. When quantized to the configuration  $(b_l, b_e) = (1, 32)$ , BitTransformer requires 4 times less memory than its full-precision counterpart to store its parameters. BitTransGNN additionally contains GNN parameters on top of the BitTransformer model parameters; however, despite the additional overhead, BitTransGNN only has a 6% higher cost than BitTransformer. The additional overhead is highly insignificant compared to the advantage obtained through quantization. The parameter memory costs could ultimately be reduced by almost 32 times if the embedding module is also quantized to  $b_e = 1$ -bit. As BitTransGNN variants have comparable size to the BitTransformer, our framework can be trained with minor additional memory cost. Consequently, the distillation procedure in Chapter 3.5.2 can also be performed in a memory-efficient manner, since the joint BitTransGNN variants add only a minor additional parameter memory cost to that of a solitary quantized transformer.

## 4.2 Energy Efficiency

Binarization allows to replace resource-intensive multiplication operations with additions and subtractions. The ternary alternative  $\mathcal{S} = \{-1, 0, 1\}$  also has this potential. Quantization sets containing four or five states, such as the sets shown in Table 3.2, can also benefit from such a reduction by using shift operations in

Table 4.3: Number of bits required to store BitTransformer and BitTransGNN parameters in different quantization scenarios.

| Model Type            | Model Size (MB) | Model Size (Ratio)      |
|-----------------------|-----------------|-------------------------|
| <b>BitTransformer</b> |                 |                         |
| W32E32                | 438.0           | $B_{\text{BERT}}$       |
| W1E32                 | 106.2           | $0.2425B_{\text{BERT}}$ |
| W32E1                 | 345.6           | $0.7891B_{\text{BERT}}$ |
| W1E1                  | 13.8            | $0.0316B_{\text{BERT}}$ |
| <b>BitTransGNN</b>    |                 |                         |
| W32E32                | 444.5           | $1.0148B_{\text{BERT}}$ |
| W1E32                 | 112.7           | $0.2572B_{\text{BERT}}$ |
| W32E1                 | 352.1           | $0.8039B_{\text{BERT}}$ |
| W1E1                  | 20.3            | $0.0463B_{\text{BERT}}$ |

binary arithmetic over elements -2 and 2 [23, 25]. Binary shifts can be conducted almost as fast as additions in most hardware architectures, hence, we treat the energy consumption of binary shift operations as equivalent to additions to simplify calculations [26].

A full-precision matrix multiplication with matrices  $\mathbf{X}_{\text{lin}} \in \mathbb{R}^{M \times d_x}$  and  $\mathbf{W}_{\text{lin}} \in \mathbb{R}^{d_x \times d_y}$  performs a total of  $M \times (d_x - 1) \times d_y$  addition operations. In a linear transformation of the form  $\mathbf{Y}_{\text{lin}} = \mathbf{X}_{\text{lin}} \mathbf{W}_{\text{lin}} + \mathbf{b}$ , the bias term adds  $M \times d_y$  more additions, totaling to  $o_{\text{add}} = M \times d_x \times d_y$  addition operations. Consequently,  $o_{\text{mult}} = M \times d_x \times d_y$  scalar multiplications are performed in a linear transformation layer.

Binarization allows one to replace the scalar multiplications related to matrix multiplication with additions and subtractions. After our quantization procedure, scalar multiplication is only used to scale the output back to its original precision in the dequantization stage. During dequantization, the linear transformation output is multiplied by the scalar factor  $\alpha_W \alpha_X$ . This operation introduces a total of  $M \times d_y$  multiplication operations. Considering addition operations; due to the -1 elements that occur within the weight matrix after quantization, binarization adds up to  $M \times d_x \times d_y$  addition operations. Note that we do not use a bias term in our quantized linear layer implementation, which removes  $M \times d_y$  additions

from the total count. As a result, the total number of addition and multiplication operations after binarization can be written as follows:

$$o_{add} = M \times (2d_x - 1) \times d_y, \quad (4.3)$$

$$o_{mult} = M \times d_y. \quad (4.4)$$

This formulation is valid for the quantization sets  $S = \{-1, 1\}$  and  $S = \{-1, 0, 1\}$ .

Calculations are slightly different for the sets  $S = \{-2, -1, 0, 1\}$  and  $S = \{-2, -1, 0, 1, 2\}$ . Multiplying by 2 is equivalent to a left shift in binary arithmetic, and multiplying by  $-2$  has a negation step following the left shift. We approximate the energy consumed by the shift operations as equal to that of additions [26]. Each  $-2$  or  $2$  element within the matrices have to be left shifted by one bit, adding  $M \times d_x \times d_y$  more addition operations in the worst case. As a result, the number of addition operations for the two higher-precision sets becomes the following:

$$o_{add} = M \times (3d_x - 1) \times d_y. \quad (4.5)$$

The count of multiplication operations is the same in all quantization scenarios. The total number of additions and multiplications carried out by each BERT-based transformer component are listed in Table 4.4 based on this formulation.

We show the exact values of the number of arithmetic operations performed by a BitTransformer that employs a base-sized BERT as its transformer model during a forward-pass for different quantization rates in Table 4.5. The count of addition operations conducted by the transformer model double with 1-bit/1.58-bit quantization, and triple with 2-bit/2.32-bit one. Meanwhile, there is a drastic decrease in the number of multiplications, as low-bit quantization reduces the count of multiplication operations to less than 3% of the amount performed by the full-precision transformer.

The changes in the number of arithmetic operations conducted by the transformer models due to quantization also alter the total energy consumption of these models. We calculate the energy consumed by the arithmetic operations within a linear transformation as following:

$$E_{lin} = o_{add} \times \hat{E}_{add} + o_{mult} \times \hat{E}_{mult}. \quad (4.6)$$

Table 4.4: Number of addition and multiplication operations performed by the transformer. Exact operation counts for different model variants can be calculated using the variables listed in Table 4.1. We use a window length  $M = 128$ .

| Additions             |                                                                                                                                                                        |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Full-Precision</b> |                                                                                                                                                                        |
| Embeddings            | $2 \times M \times d_h$                                                                                                                                                |
| Attention             | $L \times (H \times 3 \times M \times d_h \times d_s + 2 \times H \times M \times (M - 1) \times d_s + M \times d_h \times d_h + 2 \times M \times d_h)$               |
| Feed Forward Network  | $L \times (M \times d_h \times d_f + M \times d_f \times d_h + 2 \times M \times d_h)$                                                                                 |
| Pooler                | $d_h \times d_h$                                                                                                                                                       |
| Classifier            | $d_h \times d_o$                                                                                                                                                       |
| <b>1-bit/1.58-bit</b> |                                                                                                                                                                        |
| Embeddings            | $2 \times M \times d_h$                                                                                                                                                |
| Attention             | $L \times (H \times 3 \times M \times (2d_h - 1) \times d_s + 2 \times H \times M \times (M - 1) \times d_s + M \times (2d_h - 1) \times d_h + 2 \times M \times d_h)$ |
| Feed Forward Network  | $L \times (M \times (2d_h - 1) \times d_f + M \times (2d_f - 1) \times d_h + 2 \times M \times d_h)$                                                                   |
| Pooler                | $(2d_h - 1) \times d_h$                                                                                                                                                |
| Classifier            | $(2d_h - 1) \times d_o$                                                                                                                                                |
| <b>2-bit/2.32-bit</b> |                                                                                                                                                                        |
| Embeddings            | $2 \times M \times d_h$                                                                                                                                                |
| Attention             | $L \times (H \times 3 \times M \times (3d_h - 1) \times d_s + 2 \times H \times M \times (M - 1) \times d_s + M \times (3d_h - 1) \times d_h + 2 \times M \times d_h)$ |
| Feed Forward Network  | $L \times (M \times (3d_h - 1) \times d_f + M \times (3d_f - 1) \times d_h + 2 \times M \times d_h)$                                                                   |
| Pooler                | $(3d_h - 1) \times d_h$                                                                                                                                                |
| Classifier            | $(3d_h - 1) \times d_o$                                                                                                                                                |
| Multiplications       |                                                                                                                                                                        |
| <b>Full-Precision</b> |                                                                                                                                                                        |
| Embeddings            | $2 \times M \times d_h$                                                                                                                                                |
| Attention             | $L \times (H \times 3 \times M \times d_h \times d_s + 2 \times H \times M \times M \times d_s + M \times d_h \times d_h + 2 \times M \times d_h)$                     |
| Feed Forward Network  | $L \times (M \times d_h \times d_f + M \times d_f \times d_h + 2 \times M \times d_h)$                                                                                 |
| Pooler                | $d_h \times d_h$                                                                                                                                                       |
| Classifier            | $d_h \times d_o$                                                                                                                                                       |
| <b>1-bit/1.58-bit</b> |                                                                                                                                                                        |
| Embeddings            | $2 \times M \times d_h$                                                                                                                                                |
| Attention             | $L \times (H \times 3 \times M \times d_s + 2 \times H \times M \times M \times d_s + M \times d_h + 2 \times M \times d_h)$                                           |
| Feed Forward Network  | $L \times (M \times d_f + M \times d_h + 2 \times M \times d_h)$                                                                                                       |
| Pooler                | $d_h$                                                                                                                                                                  |
| Classifier            | $d_o$                                                                                                                                                                  |
| <b>2-bit/2.32-bit</b> |                                                                                                                                                                        |
| Embeddings            | $2 \times M \times d_h$                                                                                                                                                |
| Attention             | $L \times (H \times 3 \times M \times d_s + 2 \times H \times M \times M \times d_s + M \times d_h + 2 \times M \times d_h)$                                           |
| Feed Forward Network  | $L \times (M \times d_f + M \times d_h + 2 \times M \times d_h)$                                                                                                       |
| Pooler                | $d_h$                                                                                                                                                                  |
| Classifier            | $d_o$                                                                                                                                                                  |

where  $E_{\text{lin}}$  is the total energy required to perform the linear transformation, and  $\hat{E}_{\text{add}}$  and  $\hat{E}_{\text{mult}}$  refer to the unit energy required to perform a scalar addition and a scalar multiplication operation, respectively. Information on the unit energy consumption of addition and multiplication operations is listed in Table 4.6 for different data types [104, 105]. Additions consume considerably less energy compared to multiplications with full-precision data, and the advantage becomes even more significant when integer data types are utilized. Since we quantize activations to 8-bits and use binary weights, our architecture has the potential to utilize 8-bit integers if coding frameworks are adapted to handle such data types.

Table 4.5: Count of additions and multiplications, and the ratio of arithmetic operations for BitTransformer models against their full-precision counterparts.

| Bit-Precision  | Additions      |       | Multiplications |       |
|----------------|----------------|-------|-----------------|-------|
|                | Count          | Ratio | Count           | Ratio |
| 32-bit         | 11,176,786,944 | 1.000 | 11,179,146,240  | 1.000 |
| 1-bit/1.58-bit | 22,038,410,476 | 1.972 | 317,522,708     | 0.028 |
| 2-bit/2.32-bit | 32,910,651,628 | 2.945 | 317,522,708     | 0.028 |

Table 4.6: Unit energy consumption of addition and multiplication operations listed for 7 nm and 45 nm processors, and the energy ratio between the two operations for three different data types.

|                       | 7nm Energy (pJ) |       |       | 45nm Energy (pJ) |      |       |
|-----------------------|-----------------|-------|-------|------------------|------|-------|
|                       | Mul             | Add   | Ratio | Mul              | Add  | Ratio |
| 8-bit Integer         | 0.07            | 0.007 | 10.0  | 0.2              | 0.03 | 6.67  |
| 32-bit Integer        | 1.48            | 0.03  | 49.3  | 3.1              | 0.1  | 31.0  |
| 32-bit Floating-Point | 1.31            | 0.38  | 3.45  | 3.7              | 0.9  | 4.11  |

We list the theoretical energy consumed per epoch by the arithmetic operations of the baseline methods and the BERT-based BitTransGNN variants in Table 4.7 for various bit-precisions and data types. During the calculation of the energy consumption of BitTransGNN and BitTransformer variants, we also took the differences in the way transformer and GNN models process information into account. GNN processes the information through the entire graph at each pass, while BitTransformer only processes a mini-batch of the documents provided to the model. Due to this difference in their characteristics, we inspect the total number of operations conducted by the two models per epoch rather than per iteration while evaluating the energy expense of BitTransformer and BitTransGNN variants to allow for a fair comparison between the two models. Note that energy costs are dataset-dependent. To outline the worst case, we base our analysis on the 20NG dataset, which produces the largest cost among the alternatives we consider.

When processed with 32-bit floating-point precision, transformer quantization to 1-bit/1.58-bit representations can theoretically halve the energy consumption, and 2-bit/2.32-bit quantization can reduce the consumption by more than 1.4 times. BitTransGNN<sub>Static</sub> can be utilized with negligible additional energy cost compared to quantized transformer baselines, as it only employs the GNN once per epoch. On the other hand, the dynamic training scheme utilizes the GNN after every batch update, leading to an increase in the model’s energy use since the GNN has to operate over the full graph structure at each execution. For this reason, BitTransGNN<sub>Dynamic</sub> gains less benefit from the potential advantages in energy consumption that can be achieved through quantization. The use of BitTransGNN<sub>Dynamic</sub> requires about 14% more energy expense compared to the use of BitTransGNN<sub>Static</sub>. Despite this additional overhead, all BitTransGNN<sub>Dynamic</sub> variants require less energy than full-precision baselines. Quantizing activations to 8-bit integers as demonstrated in Chapter 3.1.1 theoretically makes it possible to process the data with 8-bit integers. If the transformers are coded to handle such data types, integer-type representations offer much more significant potential in reducing energy costs, as 1-bit quantization can reduce energy consumption by more than 16 times when using 32-bit integers and by more than 100 times if the data are processed with 8-bit integers.

Table 4.7: Energy consumption by arithmetic operations per epoch for baseline quantized transformer models and BitTransGNN variants over 7nm processors. Ratios of the energy overhead of each instance against that of the full-precision transformer baseline are also listed alongside the exact values of energy expense.

| Bit-Precision  | Data Type | BitTransformer (J)                 |         |         |       |
|----------------|-----------|------------------------------------|---------|---------|-------|
|                |           | Add                                | Mul     | Total   | Ratio |
| 1-bit/1.58-bit | float32   | 157.828                            | 7.839   | 165.667 | 0.465 |
| 2-bit/2.32-bit |           | 235.689                            | 7.839   | 243.528 | 0.684 |
| 32-bit         |           | 80.042                             | 275.994 | 356.036 | 1.000 |
| 1-bit/1.58-bit | int32     | 12.460                             | 8.856   | 21.316  | 0.060 |
| 2-bit/2.32-bit |           | 18.607                             | 8.856   | 27.463  | 0.077 |
| 1-bit/1.58-bit | int8      | 2.907                              | 0.419   | 3.326   | 0.009 |
| 2-bit/2.32-bit |           | 4.342                              | 0.419   | 4.761   | 0.013 |
| Bit-Precision  | Data Type | BitTransGNN <sub>static</sub> (J)  |         |         |       |
|                |           | Add                                | Mul     | Total   | Ratio |
| 1-bit/1.58-bit | float32   | 157.847                            | 7.905   | 165.752 | 0.466 |
| 2-bit/2.32-bit |           | 235.708                            | 7.905   | 243.613 | 0.684 |
| 32-bit         |           | 80.061                             | 276.060 | 356.121 | 1.000 |
| 1-bit/1.58-bit | int32     | 12.462                             | 8.931   | 21.392  | 0.060 |
| 2-bit/2.32-bit |           | 18.609                             | 8.931   | 27.539  | 0.077 |
| 1-bit/1.58-bit | int8      | 2.908                              | 0.422   | 3.301   | 0.009 |
| 2-bit/2.32-bit |           | 4.342                              | 0.422   | 4.764   | 0.013 |
| Bit-Precision  | Data Type | BitTransGNN <sub>dynamic</sub> (J) |         |         |       |
|                |           | Add                                | Mul     | Total   | Ratio |
| 1-bit/1.58-bit | float32   | 169.079                            | 46.663  | 215.742 | 0.606 |
| 2-bit/2.32-bit |           | 246.941                            | 46.663  | 293.603 | 0.825 |
| 32-bit         |           | 91.294                             | 314.817 | 406.111 | 1.140 |
| 1-bit/1.58-bit | int32     | 13.348                             | 52.718  | 66.067  | 0.186 |
| 2-bit/2.32-bit |           | 19.495                             | 52.718  | 72.214  | 0.203 |
| 1-bit/1.58-bit | int8      | 3.115                              | 2.493   | 5.608   | 0.016 |
| 2-bit/2.32-bit |           | 4.549                              | 2.493   | 7.042   | 0.020 |

## Chapter 5

# Experimental Results and Evaluations

In this chapter, we present our experimental setup and display the results of our study. We list the datasets used to train BitTransGNN variants in Chapter 5.1. We display the ranges of hyperparameters that we search through in Chapter 5.2. We outline our experimental setup in Chapter 5.3. In Chapter 5.4, we present the results of BitTransGNN and its inductive variants, and compare these methods with full-precision and quantized transformer alternatives.

### 5.1 Datasets

We assessed our method on various language tasks, including sentence topic classification, sentiment analysis, textual entailment and semantic similarity. We conducted our experiments over nine benchmark datasets. We used *20 News-groups (20NG)*, *Movie Review (MR)*, *R8*, *R52* and *Ohsumed* datasets, as well as the *Recognizing Textual Entailment (RTE)*, *Semantic Textual Similarity Benchmark (STS-B)*, *Corpus of Linguistic Acceptability (CoLA)*, and *Microsoft Research Paraphrase Corpus (MRPC)* tasks from the *GLUE* benchmark [106]. We

used the original validation set as a replacement for the test set for RTE, STS-B and CoLA datasets, as the original test set did not contain any labels. We downloaded the GLUE datasets from the HuggingFace library [79]. The rest of the datasets were retrieved from TextGCN’s repository.<sup>1</sup>

The detailed explanations of the datasets we use are given below:

**20NG dataset** contains 18,846 documents that are partitioned evenly across 20 different newsgroups. The dataset is split to 11,314 training and 7,532 test documents.

**MR dataset** contains single sentence movie reviews on which we apply sentiment classification. The dataset contains a total of 10,662 documents, split equally to positive and negative reviews. The documents are split to 7,108 training documents and 3,554 test documents.

**R8 (all terms version) and R52 datasets** are subsets of the Reuters 21,578 dataset. R8 dataset has 8 classes, while R52 contains 52 different categories. R8 has 7,674 documents split to 5,485 training and 2,189 test documents, while R52 contains 9,100 documents separated to 6,532 training and 2,568 test documents.

**Ohsumed dataset** contains medical abstracts extracted from important medical literature in the MEDLINE database. It contains the abstracts of 13,329 unique cardiovascular diseases extracted from the first 20,000 papers in 1991. Each document is linked to one or more associated categories among 23 diseases. We focused on single-label data instances and eliminated the documents that are linked to multiple labels, leaving a total of 7,400 documents. These documents are split to 3,357 training documents and 4,043 test documents.

**RTE dataset** contains sentence pairs and labels on whether the two sentences are entailed or not. We eliminated the test set, since it does not contain any labels, and used the validation set for that purpose. As a results, the dataset includes 2,490 training pairs and 277 test pairs.

---

<sup>1</sup>[https://github.com/yao8839836/text\\_gcn](https://github.com/yao8839836/text_gcn)

**STS-B dataset** contains a collection of sentence pairs drawn from different sources and human annotated similarity scores between 0 and 5.

**MRPC dataset** contains sentence pairs that are extracted from news sources and human annotations on whether the pairs are semantically equivalent. We replaced the test set with the validation set as in the RTE dataset. The dataset contains 3,668 training pairs and 408 test pairs.

**CoLA dataset** contains sentences extracted from books and journals. Each sentence is annotated on whether it is a grammatically correct English sentence. It comprises 5,749 training pairs and 1,500 test pairs.

In STS-B, we perform a regression task on sentence-pair similarity scores, whereas we apply text classification on the rest of the datasets. We list the statistics of the datasets used in our experiments in Table 5.1.

Table 5.1: Statistics and information related to datasets.

|                 | 20NG       | MR        | R8        | R52       | Ohsumed   | RTE       | STS-B     | CoLA    | MRPC      |
|-----------------|------------|-----------|-----------|-----------|-----------|-----------|-----------|---------|-----------|
| # Words         | 42,757     | 18,764    | 7,688     | 8,892     | 14,157    | 16,096    | 13,582    | 5,704   | 15,654    |
| # Docs          | 18,846     | 10,662    | 7,674     | 9,100     | 7,400     | 2,767     | 7,249     | 9,594   | 5,801     |
| # Training Docs | 11,314     | 7,108     | 5,485     | 6,532     | 3,357     | 2,490     | 5,749     | 8,551   | 4,076     |
| # Test Docs     | 7,532      | 3,554     | 2,189     | 2,568     | 4,043     | 277       | 1,500     | 1,043   | 1,725     |
| # Graph Nodes   | 61,603     | 29,426    | 15,362    | 17,992    | 21,557    | 18,586    | 20,831    | 15,298  | 21,455    |
| # Graph Edges   | 24,671,120 | 1,701,424 | 3,165,430 | 3,981,246 | 7,456,448 | 2,085,620 | 1,195,999 | 304,885 | 1,737,541 |
| # Classes       | 20         | 2         | 8         | 52        | 23        | 2         | 1 (N/A)   | 2       | 2         |
| Average Length  | 221.26     | 20.39     | 65.72     | 69.82     | 135.82    | 52.2      | 20.5      | 7.7     | 43.8      |
| Max Length      | 819        | 39        | 520       | 612       | 476       | 244       | 99        | 42      | 81        |

The preprocessing of 20NG, MR, R8, R52 and Ohsumed datasets follows [83]. In GLUE datasets, we tokenized the sentences with the BERT tokenizer and placed the relevant tokens to the text graph nodes.

## 5.2 Hyperparameter Search

We perform a hyperparameter search for  $\lambda$ , batch size, and  $d_g$  in the BitTrans-GNN methods. We use the model checkpoint of the best-performing instance over the validation set when conducting the inductive methods DS and KD over

BitTransGNN. We search for the best-performing  $\alpha_d$  and  $\tau$  on the validation set to be used in the distillation process. We list the range of hyperparameters through which we conduct the hyperparameter search in Table 5.2.

Table 5.2: Parameter sets that are swept during hyperparameter search.

|                                        | Min | Max | Step |
|----------------------------------------|-----|-----|------|
| <b>Batch size</b>                      | 32  | 128 | 32   |
| $\lambda$                              | 0.0 | 1.0 | 0.05 |
| <b>Dropout</b>                         | 0.0 | 1.0 | 0.1  |
| <b>GNN hidden layer size</b> ( $d_g$ ) | 32  | 256 | 32   |
| $\alpha_d$                             | 0.0 | 1.0 | 0.1  |
| <b>Temperature</b> ( $\tau$ )          | 0.5 | 5.0 | 0.25 |

### 5.3 Experimental Setup

We integrate base-sized pre-trained BERT and RoBERTa models available in the Hugging Face Transformers library [79] into the BitTransGNN architecture. We perform experiments with transformers quantized to 1-bit, 1.58-bit, and 2.32-bit parameter precision, using the corresponding quantization sets listed in Table 3.2. We quantize and fine-tune the pre-trained transformer models before integrating them into BitTransGNN following the procedure outlined in Chapter 3.1.3. We employ QAFT BitTransformer variants in BitTransGNN<sub>Static</sub> methods. We investigate the use of both the QAFT and PFTQ BitTransGNN variants in BitTransGNN<sub>Dynamic</sub> and choose the method that performs best on the validation set. In knowledge distillation, we utilize the PFTQ BitTransformer variants as the student model.

We compare our methods against full-precision and quantized transformer models. We use full-precision BERT [3] and RoBERTa [4] transformers, as well as their quantized variants that are fine-tuned in isolation without being integrated into the BitTransGNN architecture, as our baseline models. We use QAFT to fine-tune the baseline quantized transformer models. We group quantized model

instances under the BitTransformer label while displaying their results and indicate the bit-precision of their parameters alongside in parentheses. We also report the performance of BERTGCN model variants from [85] on the *20NG*, *MR*, *R8*, *R52* and *Ohsumed* datasets. Note that we treat these model instances as a full-precision baseline that jointly employs a full-precision transformer along with a GCN model. We do not make a direct comparison with these variants and rather aim to show their position in the efficiency-performance spectrum by displaying their results.

We trained the model for a maximum of 100 epochs using the Adam optimizer. We used early stopping if the validation metric did not improve for 10 consecutive epochs. All models were developed using the PyTorch library [107]. We ran our experiments with five different seeds and reported the mean and standard deviation of our results. To find the optimal hyperparameters, we performed a hyperparameter search on the validation set, which contains 10% of the original training set.

## 5.4 Results and Evaluations

We present the results of the transductive and inductive BitTransGNN methods trained on the listed language tasks and compare their performance against the baseline models in Tables 5.3 and 5.4. We refer to the BitTransGNN<sub>Static</sub> and the BitTransGNN<sub>Dynamic</sub> variants by the name of the transformer model integrated into the architecture and by the bit-precision of the transformer parameters. We refer to the inductive BitTransGNN methods DS and KD as BitTransGNN-DS and BitTransGNN-KD, respectively, along with the transformer type and parameter bit-precision. We apply DS only on BitTransGNN<sub>Dynamic</sub> model instances, as BitTransGNN<sub>Static</sub> involves no quantized transformer training. In KD, we choose the best-performing model instance from BitTransGNN<sub>Static</sub> and BitTransGNN<sub>Dynamic</sub> on the validation set as the teacher model. We use a BitTransformer quantized with PFTQ to the same bit-precision as the teacher BitTransGNN as the student model. To assess model performances, we report

Matthew’s Correlation metric for the CoLA dataset and the Pearson Correlation metric for the STS-B dataset. For the rest of the tasks, we report the accuracy of each method on the corresponding language task.

#### 5.4.1 Transductive Model Performance

The performances of BitTransGNN variants and the baseline methods, along with their theoretical parameter memory cost and energy expense per epoch, are displayed in Table 5.3. BitTransGNN methods significantly outperform quantized transformer models in all language tasks. BERT-based 1-bit BitTransGNN<sub>Dynamic</sub> reports 88.65% mean accuracy in the 20NG dataset with  $\lambda = 0.55$ , 7.16% higher than its 1-bit BitTransformer counterpart. The model instance outperforms all full-precision and quantized transformer baselines on the 20NG dataset. Similarly, 1.58-bit BitTransGNN<sub>Dynamic</sub> achieves 70.08% mean accuracy with  $\lambda = 0.50$  on the Ohsumed dataset, outperforming the 1.58-bit BitTransformer by 6.00%. Besides the impressive performance, the significance of the graph module is also reflected in the optimal  $\lambda$  values used by BitTransGNN in these two instances, as both of these models obtain more than 50% of their predictions from the GNN module. Binarization leads to the most notable performance drop in GLUE tasks, especially in the CoLA and STS-B datasets, and our methods successfully restore these losses. 1.58-bit BERT-based BitTransGNN<sub>Dynamic</sub> method outperforms the 1.58-bit BitTransformer by 20.70% on the CoLA dataset and the 2.32-bit RoBERTa-based BitTransGNN<sub>Dynamic</sub> surpasses the 2.32-bit RoBERTa performance by 13.24% on the STS-B dataset. The improvements are modest for certain datasets, such as R8, as binarization leads to a minor loss in these instances.

#### 5.4.2 Inductive Model Performance

We show the performance of the inductive BitTransGNN variants and the baseline methods, and show the theoretical memory and energy overhead of each

Table 5.3: Comparison of BitTransGNN methods with quantized and full-precision transformers. The best results are boldened, the second-best models are underlined, and the third-best methods are italicized.

|                                | Size  | Energy | 20NG                | MR                  | R8                  | R52                 | Ohsumed             |
|--------------------------------|-------|--------|---------------------|---------------------|---------------------|---------------------|---------------------|
| Model                          | MB    | J      | Accuracy            | Accuracy            | Accuracy            | Accuracy            | Accuracy            |
| Transformers                   |       |        |                     |                     |                     |                     |                     |
| BERT                           | 438.0 | 356.0  | 83.00 ± 0.53        | 84.14 ± 0.84        | 97.36 ± 0.26        | 95.09 ± 0.49        | 68.42 ± 0.87        |
| RoBERTa                        | 498.6 | 356.0  | 84.34 ± 0.76        | <b>89.04 ± 0.47</b> | <i>98.12 ± 0.10</i> | <i>96.09 ± 0.11</i> | <b>71.39 ± 0.39</b> |
| BitTransformer                 |       |        |                     |                     |                     |                     |                     |
| BERT (1-bit)                   | 106.2 | 165.7  | 81.49 ± 0.46        | 76.92 ± 0.56        | 97.30 ± 0.50        | 94.42 ± 0.67        | 63.34 ± 1.00        |
| BERT (1.58-bit)                | 112.4 | 165.7  | 82.15 ± 1.14        | 77.66 ± 0.59        | 97.29 ± 0.42        | 95.43 ± 0.22        | 64.08 ± 1.25        |
| BERT (2.32-bit)                | 120.3 | 243.5  | 83.05 ± 0.41        | 81.86 ± 1.18        | 96.82 ± 0.60        | 94.85 ± 0.73        | 66.62 ± 0.59        |
| RoBERTa (1-bit)                | 166.9 | 165.7  | 79.51 ± 0.97        | 77.27 ± 0.63        | 97.27 ± 0.24        | 94.77 ± 0.47        | 61.99 ± 1.03        |
| RoBERTa (1.58-bit)             | 173.1 | 165.7  | 80.62 ± 0.68        | 79.49 ± 0.26        | 97.48 ± 0.19        | 95.62 ± 0.21        | 64.38 ± 1.84        |
| RoBERTa (2.32-bit)             | 181.0 | 243.5  | 82.37 ± 0.85        | 83.28 ± 0.40        | 97.60 ± 0.21        | 95.61 ± 0.20        | 66.08 ± 0.94        |
| BitTransGNN <sup>Static</sup>  |       |        |                     |                     |                     |                     |                     |
| BERT (1-bit)                   | 112.7 | 165.8  | 86.26 ± 0.27        | 78.64 ± 0.10        | 98.09 ± 0.02        | 94.00 ± 0.08        | 69.36 ± 0.50        |
| BERT (1.58-bit)                | 118.9 | 165.8  | 87.84 ± 0.17        | 79.02 ± 0.18        | 97.72 ± 0.06        | 95.50 ± 0.03        | 67.84 ± 0.62        |
| BERT (2.32-bit)                | 126.8 | 243.6  | 87.58 ± 0.37        | 83.24 ± 0.10        | 98.02 ± 0.04        | <u>96.12 ± 0.06</u> | <i>70.81 ± 0.49</i> |
| RoBERTa (1-bit)                | 173.3 | 165.8  | 87.67 ± 0.26        | 78.64 ± 0.12        | 97.94 ± 0.10        | <u>95.08 ± 0.03</u> | 68.12 ± 0.83        |
| RoBERTa (1.58-bit)             | 179.5 | 165.8  | 86.99 ± 0.25        | 80.48 ± 0.14        | 98.06 ± 0.12        | 95.42 ± 0.07        | 66.92 ± 0.96        |
| RoBERTa (2.32-bit)             | 187.4 | 243.6  | 87.53 ± 0.19        | 83.52 ± 0.05        | 97.70 ± 0.03        | 95.44 ± 0.03        | 70.09 ± 0.56        |
| BitTransGNN <sup>Dynamic</sup> |       |        |                     |                     |                     |                     |                     |
| BERT (1-bit)                   | 112.7 | 215.7  | <b>88.65 ± 0.08</b> | 79.27 ± 0.22        | 98.10 ± 0.11        | 95.94 ± 0.30        | 68.90 ± 1.20        |
| BERT (1.58-bit)                | 118.9 | 215.7  | <i>88.43 ± 0.24</i> | 80.23 ± 0.38        | 97.78 ± 0.29        | 95.99 ± 0.09        | 70.08 ± 0.96        |
| BERT (2.32-bit)                | 126.8 | 293.6  | <u>88.64 ± 0.19</u> | <u>85.27 ± 0.20</u> | <u>98.12 ± 0.09</u> | <b>96.26 ± 0.12</b> | <b>71.49 ± 0.44</b> |
| RoBERTa (1-bit)                | 173.3 | 215.7  | 86.46 ± 0.19        | 78.72 ± 0.51        | 97.78 ± 0.21        | 94.67 ± 0.33        | 68.30 ± 0.97        |
| RoBERTa (1.58-bit)             | 179.5 | 215.7  | 87.61 ± 0.43        | 81.07 ± 0.20        | 97.70 ± 0.14        | 95.48 ± 0.14        | 68.90 ± 1.29        |
| RoBERTa (2.32-bit)             | 187.4 | 293.6  | 87.78 ± 0.21        | <i>85.00 ± 0.32</i> | <b>98.28 ± 0.14</b> | 95.94 ± 0.34        | 70.26 ± 0.65        |
| BERTGCN Variants               |       |        |                     |                     |                     |                     |                     |
| BertGCN                        | 444.5 | 406.1  | 89.30               | 86.00               | 98.10               | 96.60               | 72.80               |
| RoBERTaGCN                     | 505.1 | 406.1  | 89.50               | 89.70               | 98.20               | 96.10               | 72.80               |
|                                | Size  | Energy | CoLA                | MRPC                | RTE                 | STS-B               |                     |
| Model                          | MB    | J      | Matthew's           | Accuracy            | Accuracy            | Pearson             |                     |
| Transformers                   |       |        |                     |                     |                     |                     |                     |
| BERT                           | 438.0 | 356.0  | <u>54.16 ± 2.65</u> | <u>74.16 ± 1.22</u> | <u>62.89 ± 4.06</u> | <u>87.32 ± 0.33</u> |                     |
| RoBERTa                        | 498.6 | 356.0  | <b>59.78 ± 1.60</b> | <b>80.19 ± 1.30</b> | <b>73.79 ± 1.87</b> | <b>89.85 ± 0.30</b> |                     |
| BitTransformer                 |       |        |                     |                     |                     |                     |                     |
| BERT (1-bit)                   | 106.2 | 165.7  | 11.73 ± 2.04        | 47.73 ± 3.44        | 50.47 ± 3.08        | 67.87 ± 1.81        |                     |
| BERT (1.58-bit)                | 112.4 | 165.7  | 15.98 ± 2.18        | 46.64 ± 2.78        | 50.98 ± 3.10        | 61.35 ± 1.67        |                     |
| BERT (2.32-bit)                | 120.3 | 243.5  | 44.77 ± 2.36        | 62.48 ± 3.16        | 53.79 ± 2.57        | 86.38 ± 0.52        |                     |
| RoBERTa (1-bit)                | 166.9 | 165.7  | 15.23 ± 2.64        | 49.76 ± 2.65        | 49.89 ± 1.00        | 43.98 ± 1.81        |                     |
| RoBERTa (1.58-bit)             | 173.1 | 165.7  | 17.07 ± 0.83        | 46.64 ± 1.96        | 50.61 ± 1.38        | 51.61 ± 1.95        |                     |
| RoBERTa (2.32-bit)             | 181.0 | 243.5  | 30.66 ± 1.86        | 47.20 ± 1.73        | 51.26 ± 1.86        | 53.51 ± 0.59        |                     |
| BitTransGNN <sup>Static</sup>  |       |        |                     |                     |                     |                     |                     |
| BERT (1-bit)                   | 112.7 | 165.8  | 12.35 ± 0.77        | 49.01 ± 0.57        | 54.01 ± 0.41        | 71.20 ± 0.04        |                     |
| BERT (1.58-bit)                | 118.9 | 165.8  | 13.67 ± 0.00        | 44.13 ± 1.08        | 53.72 ± 1.18        | 62.70 ± 0.05        |                     |
| BERT (2.32-bit)                | 126.8 | 243.6  | 44.16 ± 0.31        | 65.01 ± 0.20        | 52.49 ± 1.72        | 85.91 ± 0.01        |                     |
| RoBERTa (1-bit)                | 173.3 | 165.8  | 14.48 ± 0.00        | 50.63 ± 0.00        | 49.03 ± 0.16        | 42.15 ± 0.05        |                     |
| RoBERTa (1.58-bit)             | 179.5 | 165.8  | 18.31 ± 0.26        | 46.67 ± 0.25        | 49.31 ± 0.20        | 54.00 ± 0.13        |                     |
| RoBERTa (2.32-bit)             | 187.4 | 243.6  | 33.50 ± 0.13        | 50.03 ± 0.41        | 52.20 ± 0.32        | 54.96 ± 0.21        |                     |
| BitTransGNN <sup>Dynamic</sup> |       |        |                     |                     |                     |                     |                     |
| BERT (1-bit)                   | 112.7 | 215.7  | 28.41 ± 2.27        | 48.54 ± 1.54        | 53.43 ± 1.28        | 77.50 ± 0.47        |                     |
| BERT (1.58-bit)                | 118.9 | 215.7  | 36.68 ± 1.06        | 47.01 ± 1.41        | 52.56 ± 2.52        | 74.29 ± 0.52        |                     |
| BERT (2.32-bit)                | 126.8 | 293.6  | <i>51.28 ± 1.90</i> | <i>71.26 ± 1.22</i> | <i>61.95 ± 2.02</i> | <i>86.70 ± 0.34</i> |                     |
| RoBERTa (1-bit)                | 173.3 | 215.7  | 13.48 ± 1.28        | 51.85 ± 0.18        | 50.47 ± 2.97        | 49.52 ± 1.04        |                     |
| RoBERTa (1.58-bit)             | 179.5 | 215.7  | 18.46 ± 0.50        | 47.43 ± 0.71        | 51.84 ± 1.50        | 55.62 ± 1.21        |                     |
| RoBERTa (2.32-bit)             | 187.4 | 293.6  | 39.08 ± 0.97        | 50.39 ± 0.69        | 51.91 ± 0.69        | 66.75 ± 1.95        |                     |

model for comparison in Table 5.4. Inductive BitTransGNN variants show major improvements over the performance of quantized transformer baselines. Despite having identical inference costs, the models produced by both inductive extensions significantly outperform BitTransformer models in all tasks. DS improves quantized transformer performance with very few exceptions. Taking advantage of the additional training, KD excels even beyond DS in almost all tasks and consistently improves quantized transformer training. Both methods manage to contain a notable amount of BitTransGNN’s representational capabilities within a solitary quantized transformer. The models produced by DS and KD have the same number of parameters as full-precision models while losing a significant amount of representational capability due to quantization, putting them in a disadvantaged position against the full-precision baselines. Despite this disadvantage, inductive BitTransGNN variants show competitive performance against full-precision models in most tasks.

### 5.4.3 Efficiency-Performance Analysis

We provide a visual comparison of the model performance and the theoretical memory and energy expenses of BitTransGNN variants and baseline models on the 20NG, MR, and Ohsumed datasets in Fig. 5.1. We also include the BERT-GCN model from [85] in these plots and treat it as a full-precision reference for the joint collaboration of a transformer and a GNN.

Comparing the performance of each model against their memory consumption, we observe that the quantized models are generally placed in the left corner of the plot, as they require less memory than full-precision alternatives. BitTransGNN variants are placed higher along the vertical axis, as they generally outperform BitTransformer models and either outperform or perform comparable to full-precision models. The KD and DS methods are placed in the mid-left corner, as they lose some performance while trying to encapsulate the knowledge of the joint BitTransGNN model within a solitary quantized transformer. Despite this loss, they still perform close to the full-precision transformer model while requiring

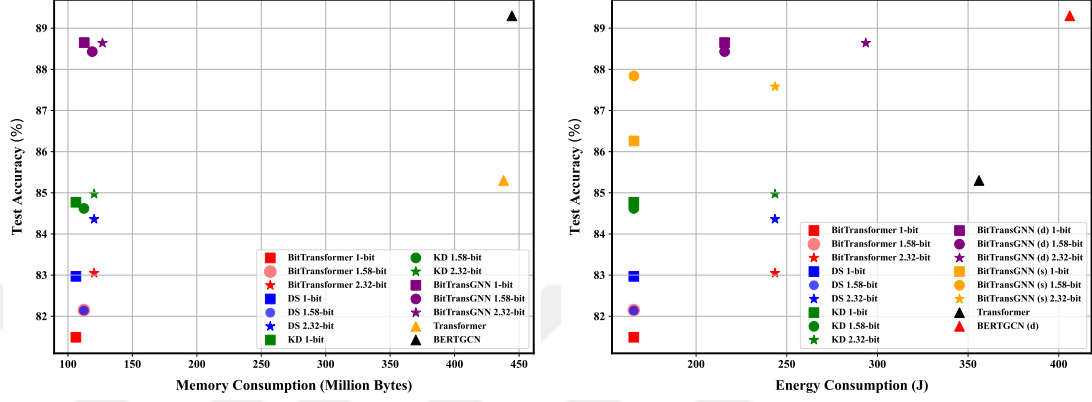
Table 5.4: Comparison of inductive BitTransGNN variants against full-precision and quantized transformer baselines. The best results are boldened, the second-best models are underlined, and the third-best methods are italicized.

|                    | Size  | Energy | 20NG                               | MR                                 | R8                                 | R52                                | Ohsumed                            |
|--------------------|-------|--------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| Model              | MB    | J      | Accuracy                           | Accuracy                           | Accuracy                           | Accuracy                           | Accuracy                           |
| Transformers       |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT               | 438.0 | 356.0  | 83.00 $\pm$ 0.53                   | 84.14 $\pm$ 0.84                   | 97.36 $\pm$ 0.26                   | 95.09 $\pm$ 0.49                   | 68.42 $\pm$ 0.87                   |
| RoBERTa            | 498.6 | 356.0  | 84.34 $\pm$ 0.76                   | <b>89.04 <math>\pm</math> 0.47</b> | <b>98.12 <math>\pm</math> 0.10</b> | <i>96.09 <math>\pm</math> 0.11</i> | <b>71.39 <math>\pm</math> 0.39</b> |
| BitTransformer     |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT (1-bit)       | 106.2 | 165.7  | 81.49 $\pm$ 0.46                   | 76.92 $\pm$ 0.56                   | 97.30 $\pm$ 0.50                   | 94.42 $\pm$ 0.67                   | 63.34 $\pm$ 1.00                   |
| BERT (1.58-bit)    | 112.4 | 165.7  | 82.15 $\pm$ 1.14                   | 77.66 $\pm$ 0.59                   | 97.29 $\pm$ 0.42                   | 95.43 $\pm$ 0.22                   | 64.08 $\pm$ 1.25                   |
| BERT (2.32-bit)    | 120.3 | 243.5  | 83.05 $\pm$ 0.41                   | 81.86 $\pm$ 1.18                   | 96.82 $\pm$ 0.60                   | 94.85 $\pm$ 0.73                   | 66.62 $\pm$ 0.59                   |
| RoBERTa (1-bit)    | 166.9 | 165.7  | 79.51 $\pm$ 0.97                   | 77.27 $\pm$ 0.63                   | 97.27 $\pm$ 0.24                   | 94.77 $\pm$ 0.47                   | 61.99 $\pm$ 1.03                   |
| RoBERTa (1.58-bit) | 173.1 | 165.7  | 80.62 $\pm$ 0.68                   | 79.49 $\pm$ 0.26                   | 97.48 $\pm$ 0.19                   | 95.62 $\pm$ 0.21                   | 64.38 $\pm$ 1.84                   |
| RoBERTa (2.32-bit) | 181.0 | 243.5  | 82.37 $\pm$ 0.85                   | 83.28 $\pm$ 0.40                   | 97.60 $\pm$ 0.21                   | 95.61 $\pm$ 0.20                   | 66.08 $\pm$ 0.94                   |
| BitTransGNN-DS     |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT (1-bit)       | 106.2 | 165.7  | 82.97                              | 77.12                              | <u>97.94</u>                       | 95.64                              | 66.41                              |
| BERT (1.58-bit)    | 112.4 | 165.7  | 82.14                              | 78.14                              | 97.76                              | 95.99                              | 68.09                              |
| BERT (2.32-bit)    | 120.3 | 243.5  | 84.36                              | <i>84.78</i>                       | 97.67                              | 96.07                              | 55.48                              |
| RoBERTa (1-bit)    | 166.9 | 165.7  | 70.72                              | 73.86                              | 96.76                              | 94.63                              | 60.77                              |
| RoBERTa (1.58-bit) | 173.1 | 165.7  | 76.27                              | 79.09                              | 97.67                              | 94.78                              | 63.49                              |
| RoBERTa (2.32-bit) | 181.0 | 243.5  | 81.69                              | 84.72                              | 97.85                              | <b>96.30</b>                       | 44.89                              |
| BitTransGNN-KD     |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT (1-bit)       | 106.2 | 165.7  | <u>84.77 <math>\pm</math> 0.23</u> | 78.47 $\pm$ 0.45                   | 97.32 $\pm$ 0.12                   | 95.81 $\pm$ 0.14                   | 69.46 $\pm$ 0.30                   |
| BERT (1.58-bit)    | 112.4 | 165.7  | <i>84.62 <math>\pm</math> 0.31</i> | 79.63 $\pm$ 0.49                   | 97.25 $\pm$ 0.37                   | 95.84 $\pm$ 0.21                   | 69.34 $\pm$ 0.13                   |
| BERT (2.32-bit)    | 120.3 | 243.5  | <b>84.97 <math>\pm</math> 0.24</b> | 84.67 $\pm$ 0.31                   | 97.80 $\pm$ 0.10                   | 95.91 $\pm$ 0.13                   | <i>70.45 <math>\pm</math> 0.26</i> |
| RoBERTa (1-bit)    | 166.9 | 165.7  | 79.24 $\pm$ 0.91                   | 77.21 $\pm$ 0.09                   | 97.54 $\pm$ 0.12                   | 94.74 $\pm$ 0.29                   | 62.65 $\pm$ 0.63                   |
| RoBERTa (1.58-bit) | 173.1 | 165.7  | 81.94 $\pm$ 0.59                   | 80.69 $\pm$ 0.25                   | 97.33 $\pm$ 0.34                   | 95.73 $\pm$ 0.08                   | 67.39 $\pm$ 0.59                   |
| RoBERTa (2.32-bit) | 181.0 | 243.5  | 83.60 $\pm$ 0.29                   | <u>85.05 <math>\pm</math> 0.50</u> | <i>97.90 <math>\pm</math> 0.11</i> | <u>96.13 <math>\pm</math> 0.20</u> | <u>70.82 <math>\pm</math> 0.33</u> |
|                    | Size  | Energy | CoLA                               | MRPC                               | RTE                                | STS-B                              |                                    |
| Model              | MB    | J      | Matthew's                          | Accuracy                           | Accuracy                           | Accuracy                           | Pearson                            |
| Transformers       |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT               | 438.0 | 356.0  | <u>54.16 <math>\pm</math> 2.65</u> | <u>74.16 <math>\pm</math> 1.22</u> | <u>62.89 <math>\pm</math> 4.06</u> | <u>87.32 <math>\pm</math> 0.33</u> |                                    |
| RoBERTa            | 498.6 | 356.0  | <b>59.78 <math>\pm</math> 1.60</b> | <b>80.19 <math>\pm</math> 1.30</b> | <b>73.79 <math>\pm</math> 1.87</b> | <b>89.85 <math>\pm</math> 0.30</b> |                                    |
| BitTransformer     |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT (1-bit)       | 106.2 | 165.7  | 11.73 $\pm$ 2.04                   | 47.73 $\pm$ 3.44                   | 50.47 $\pm$ 3.08                   | 67.87 $\pm$ 1.81                   |                                    |
| BERT (1.58-bit)    | 112.4 | 165.7  | 15.98 $\pm$ 2.18                   | 46.64 $\pm$ 2.78                   | 50.98 $\pm$ 3.10                   | 61.35 $\pm$ 1.67                   |                                    |
| BERT (2.32-bit)    | 120.3 | 243.5  | 44.77 $\pm$ 2.36                   | 62.48 $\pm$ 3.16                   | 53.79 $\pm$ 2.57                   | 86.38 $\pm$ 0.52                   |                                    |
| RoBERTa (1-bit)    | 166.9 | 165.7  | 15.23 $\pm$ 2.64                   | 49.76 $\pm$ 2.65                   | 49.89 $\pm$ 1.00                   | 43.98 $\pm$ 1.81                   |                                    |
| RoBERTa (1.58-bit) | 173.1 | 165.7  | 17.07 $\pm$ 0.83                   | 46.64 $\pm$ 1.96                   | 50.61 $\pm$ 1.38                   | 51.61 $\pm$ 1.95                   |                                    |
| RoBERTa (2.32-bit) | 181.0 | 243.5  | 30.66 $\pm$ 1.86                   | 47.20 $\pm$ 1.73                   | 51.26 $\pm$ 1.86                   | 53.51 $\pm$ 0.59                   |                                    |
| BitTransGNN-DS     |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT (1-bit)       | 106.2 | 165.7  | 29.23                              | 46.38                              | 49.10                              | 75.93                              |                                    |
| BERT (1.58-bit)    | 112.4 | 165.7  | 34.67                              | 51.38                              | 53.43                              | 73.02                              |                                    |
| BERT (2.32-bit)    | 120.3 | 243.5  | 49.06                              | <i>70.46</i>                       | <i>62.09</i>                       | 86.40                              |                                    |
| RoBERTa (1-bit)    | 166.9 | 165.7  | 12.61                              | 32.12                              | 52.35                              | 42.52                              |                                    |
| RoBERTa (1.58-bit) | 173.1 | 165.7  | 17.75                              | 41.32                              | 52.35                              | 50.13                              |                                    |
| RoBERTa (2.32-bit) | 181.0 | 243.5  | 33.46                              | 42.56                              | 55.23                              | 58.68                              |                                    |
| BitTransGNN-KD     |       |        |                                    |                                    |                                    |                                    |                                    |
| BERT (1-bit)       | 106.2 | 165.7  | 28.35 $\pm$ 2.45                   | 49.23 $\pm$ 1.79                   | 51.77 $\pm$ 1.82                   | 76.94 $\pm$ 0.84                   |                                    |
| BERT (1.58-bit)    | 112.4 | 165.7  | 35.03 $\pm$ 2.91                   | 46.83 $\pm$ 2.26                   | 51.05 $\pm$ 1.94                   | 74.05 $\pm$ 0.61                   |                                    |
| BERT (2.32-bit)    | 120.3 | 243.5  | <i>52.18 <math>\pm</math> 1.32</i> | 70.14 $\pm$ 1.47                   | 58.84 $\pm$ 2.15                   | <i>86.71 <math>\pm</math> 0.16</i> |                                    |
| RoBERTa (1-bit)    | 166.9 | 165.7  | 11.75 $\pm$ 3.46                   | 46.58 $\pm$ 1.67                   | 51.62 $\pm$ 1.05                   | 44.27 $\pm$ 1.99                   |                                    |
| RoBERTa (1.58-bit) | 173.1 | 165.7  | 20.93 $\pm$ 0.61                   | 48.87 $\pm$ 0.95                   | 48.74 $\pm$ 0.77                   | 56.05 $\pm$ 1.31                   |                                    |
| RoBERTa (2.32-bit) | 181.0 | 243.5  | 36.05 $\pm$ 2.04                   | 48.33 $\pm$ 1.25                   | 51.84 $\pm$ 2.83                   | 62.50 $\pm$ 1.21                   |                                    |

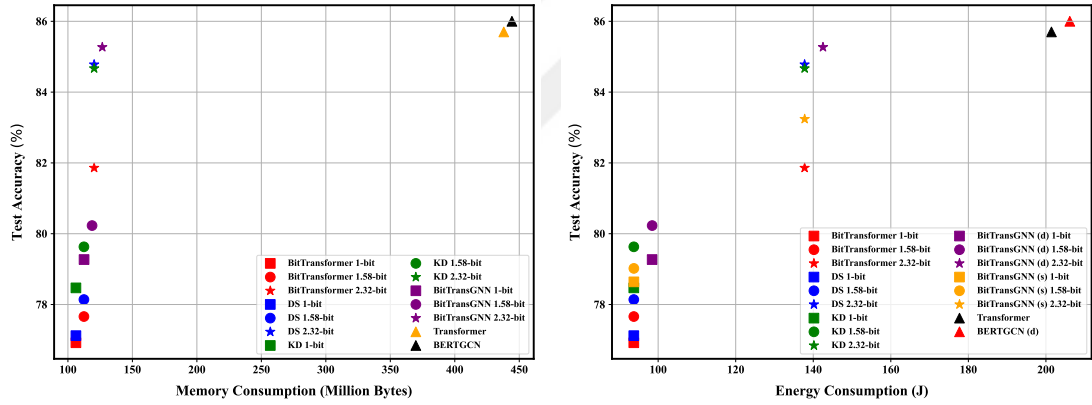
much less memory.

Inspecting the performance of each model against their energy consumptions per epoch, we observe that transformer models quantized to 1-bit, 1.58-bit, and 2.32-bit precision consume significantly less energy during inference. DS and KD methods applied on BitTransGNN variants can benefit from this energy reduction in inference time while achieving a performance comparable to that of full-precision transformers. BitTransGNN<sub>Static</sub> requires negligible additional energy costs compared to BitTransformer models while displaying much superior performance. Despite also significantly outperforming quantized transformers, BitTransGNN<sub>Dynamic</sub> consumes more energy than BitTransGNN<sub>Static</sub>. The energy overhead of the 2.32-bit BitTransGNN<sub>Dynamic</sub> model variant is closer to the full-precision transformer baseline compared to other quantized model instances. This reflects a drawback of the dynamic training scheme that arises from the utilization of the GNN model in every batch update. Nonetheless, all BitTransGNN<sub>Dynamic</sub> models still consume less energy than full-precision transformers in all datasets. By comparing the graphs of model performance against energy consumption across datasets, it can be observed that the additional overhead of the dynamic training method is clearly visible only on the 20NG dataset, which outlines the worst scenario in model efficiency due to dataset characteristics. For the MR and Ohsumed datasets, BitTransGNN<sub>Dynamic</sub> variants only require a small amount of additional energy overhead on top of that of the BitTransGNN<sub>Static</sub> model. We further discuss the implications of this shortcoming and explore potential improvements in Chapter 6.4.

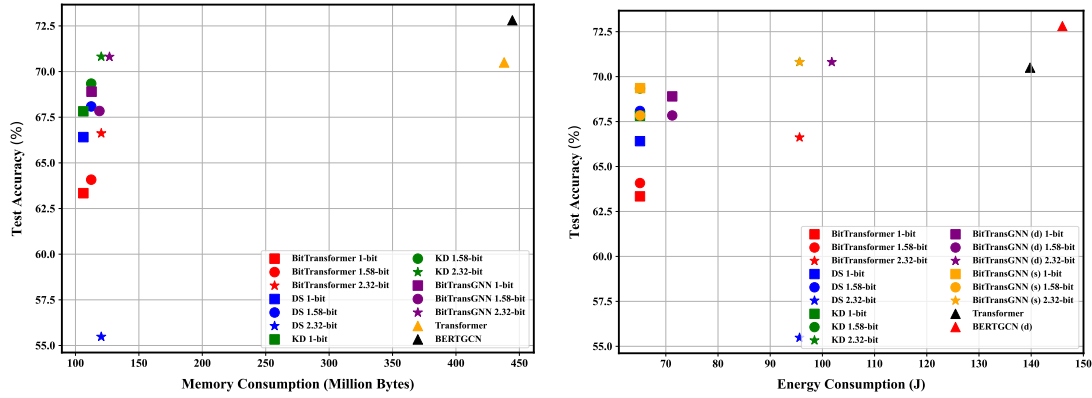
In our analysis, we focus mainly on cases in which the data is processed through floating point precision, i.e., the float32 data type. To further extend our analysis on energy efficiency to settings where neural models process input through more efficient data types, we evaluated the theoretical scenario where the data is processed through integer-type representations. We display the performance of the BitTransGNN models and baseline methods against their energy consumption when integer data representations are used in Fig. 5.2. Note that the full-precision transformer and BERTGCN (d) models shown in these plots still use float32 data types, since they are incapable of processing data with integer data



(a) 20NG dataset



(b) MR dataset



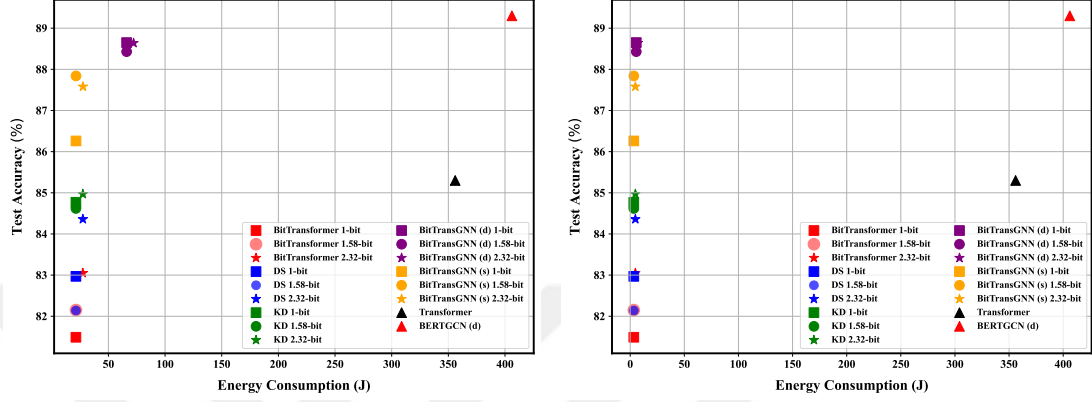
(c) Ohsumed dataset

Figure 5.1: Performance of BitTransGNN variants and baseline methods against memory consumption (left) and energy consumption (right).

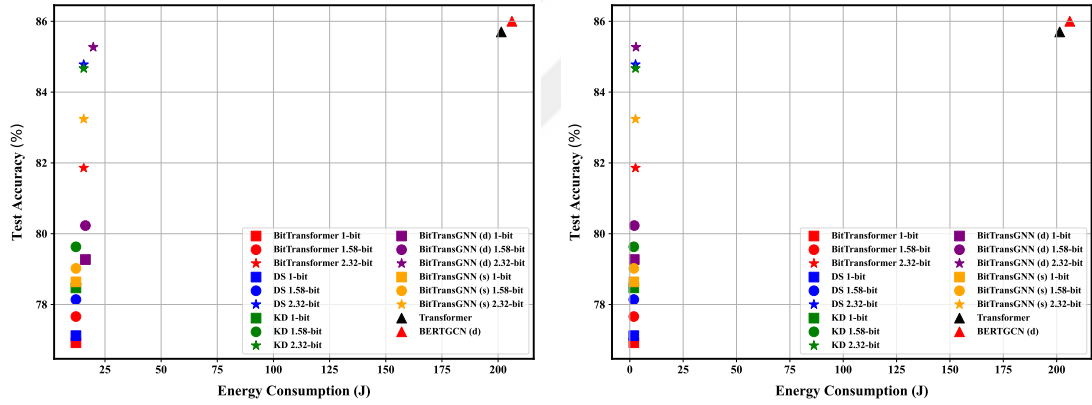
types. The efficiency advantages introduced by quantization are even more evident in these scenarios. In all tasks, the use of integer-type representations shows significant potential to reduce the energy expense of quantized transformer models compared to the use of floating-point data representations. BitTransGNN variants show great promise in leveraging this potential without having to make significant sacrifices from model capabilities.

#### 5.4.4 T-SNE Plots

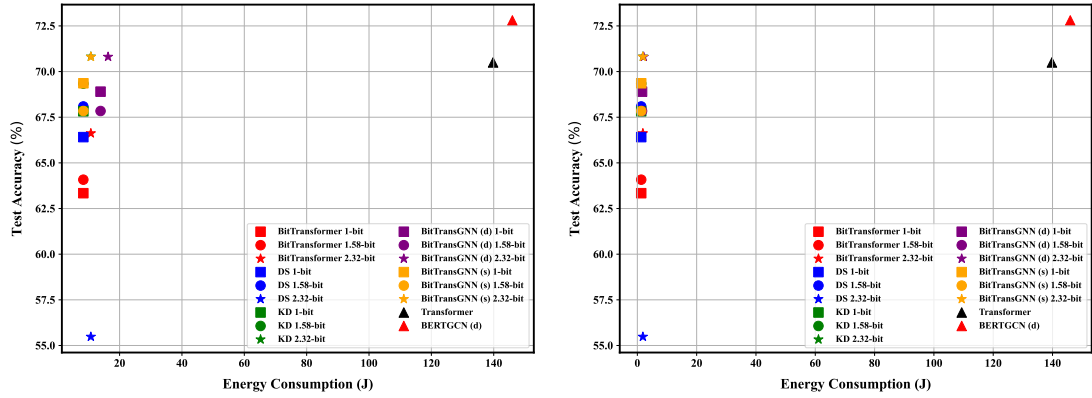
To visually analyze the effectiveness of our framework, we project the outputs of 1.0-bit BitTransGNN<sub>Dynamic</sub> and BitTransGNN-KD methods trained on the 20NG dataset onto a two-dimensional space using the t-distributed Stochastic Neighbor Embedding (t-SNE) method. We display the results in Fig. 5.3. Despite having differences in the cluster structures they construct, both methods manage to successfully group different classes into separate groups with minimal overlap between clusters, displaying the effectiveness of both the transductive and inductive methods we propose in classification tasks.



(a) 20NG dataset



(b) MR dataset



(c) Ohsumed dataset

Figure 5.2: Performance of BitTransGNN and baseline models against energy consumption when int32 (left) and int8 (right) data types are utilized.

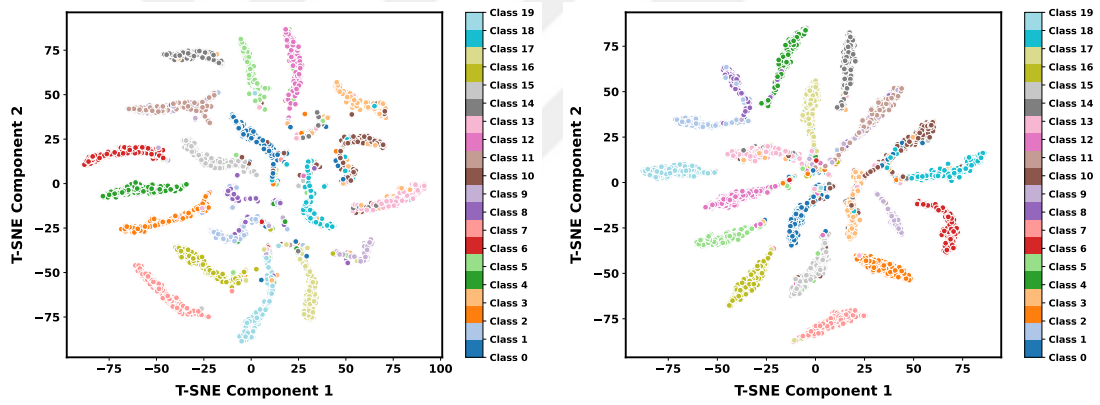


Figure 5.3: T-SNE embeddings of BitTransGNN<sub>Dynamic</sub> (left) and BitTransGNN-KD (right) outputs on the 20NG dataset displayed in a two dimensional space. Colors represent different classes predicted by the models.

# Chapter 6

## Discussion

In this chapter, we conduct a detailed analysis on the significance of our results and discuss whether they align with the assumptions that inspired the construction of our framework. We also evaluate the advantages and disadvantages regarding the construction of our model, listing potential limitations and exploring possible improvements.

### 6.1 The significance of $\lambda$

The synthesis of BitTransformer and GNN is governed by the hyperparameter  $\lambda$ , which determines the contribution of the quantized transformer and the GNN model on the final predictions of the BitTransGNN model. To investigate the factors that influence the optimal value of  $\lambda$ , we display the performance of BitTransGNN<sub>Static</sub> for different quantization rates and over varying values of  $\lambda$  on the MR and Ohsumed datasets in Fig. 6.1. The  $\lambda$  value in which the BitTransGNN variant shows the best performance is marked with an asterisk in the plots for each quantization instance.

For the MR dataset, the 1-bit model variant receives the most assistance from the GNN model, as 90% of its final outputs are received from the GNN, while the

contribution is around 30% to 50% for models quantized to higher bit-precision. Although not implying a definitive rule, these observations reveal a pattern. BitTransGNN methods employing transformers quantized to lower bit-precision benefited more from larger values of  $\lambda$ , and those that include transformers with higher bit-precision used relatively smaller values of  $\lambda$ . Hence, with increasing model capabilities, the reliance on the GNN model decreased. Therefore, BitTransGNN can adapt itself to changes in the amount of loss in the representational capability of the BitTransformer model and adjust the assistance provided to the quantized transformer from the GNN model accordingly.

Despite showing such distribution when trained on the MR dataset, BitTransGNN<sub>Static</sub> showed a different trend when trained on the Ohsumed dataset. For all quantization rates, the model shows the best performance with  $\lambda = 1.0$ , which corresponds to the scenario where the final predictions are solely determined by the GNN output and the quantized transformer only serves to initialize the document node embeddings in the text-graph. This finding displays that dataset characteristics are also significant in determining the optimal  $\lambda$ , and that there is no single  $\lambda$  value across different datasets that produces the best performance. The Ohsumed dataset can be better represented through a graph structure and the performance of the quantized transformer can be improved further by processing the data through a text-graph, while the MR dataset does not benefit as much from graph modeling compared to the Ohsumed dataset. This finding can also be observed by comparing the performance of BitTransformer and BitTransGNN variants of the same quantization rate for these datasets in Table 5.3. BitTransGNN shows a larger improvement in performance on the Ohsumed dataset, while the increase in model performance is more modest for the MR dataset in all quantization rates.

It should be noted that both graphs in Fig. 6.1 take into account the performance of BitTransGNN<sub>Static</sub>, which does not train the BitTransformer model and only optimizes the GNN parameters. In addition to the influence of the dataset characteristics, the pattern observed in the Ohsumed dataset can also be attributed to the methodology of BitTransGNN<sub>Static</sub>. Because the static training method maintains the quantized transformer in a fixed state and only updates

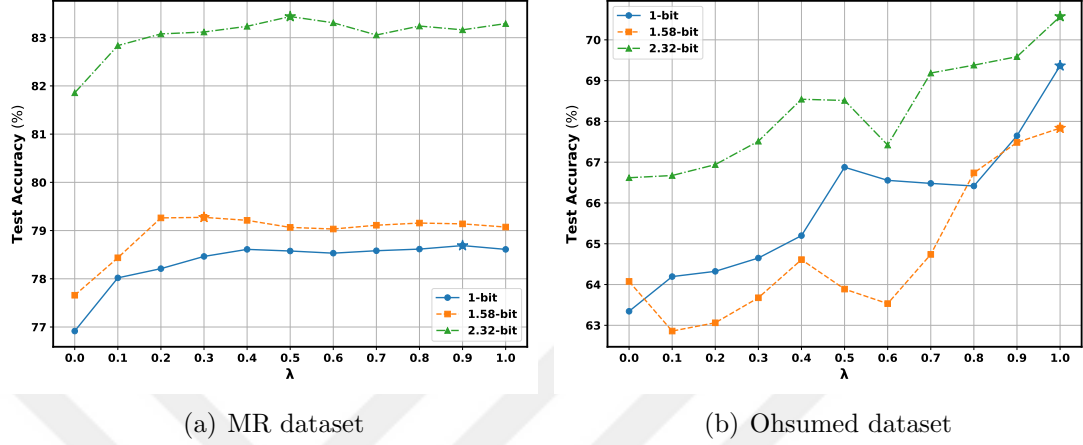


Figure 6.1: BERT-based BitTransGNN<sub>Static</sub> model performance against varying values of  $\lambda$  listed for different quantization rates.

the GNN model parameters, it may implicitly prioritize GNN’s contribution at the output stage over that of the BitTransformer model. In contrast to the behavior observed with the static training scheme, BitTransGNN<sub>Dynamic</sub> models also make use of the BitTransformer model in the output stage. The 1-bit BitTransGNN<sub>Dynamic</sub> performs the best for  $\lambda = 0.85$ , and the 1.58-bit and 2.32-bit model variants use  $\lambda = 0.50$ . Because the dynamic method updates the BitTransformer model parameters as well, a change in the relative influence of the two models is observed at the output stage, compared to the pattern seen in BitTransGNN<sub>Static</sub>.

The choice of  $\lambda$  was also found to be highly influential in the success of our inductive model variant DS. Although mostly surpassing the performance of BitTransformer models, DS showed inferior performance when implemented over the 2.32-bit BitTransGNN<sub>Dynamic</sub> trained on the Ohsumed dataset. Inspecting the optimal  $\lambda$  for these instances, we observed that both BERT- and RoBERTa-based alternatives returned the best performance when operated with  $\lambda = 1.0$ , corresponding to the case when BitTransGNN solely used the outputs retrieved from the GNN module. DS does not conduct any additional training on the quantized transformer after it is separated from the joint BitTransGNN architecture. Hence, in these scenarios, the quantized transformer is only trained to produce embeddings for the GNN model, without having a direct influence on the final

predictions. Based on the performance drop observed in these scenarios, we can comment that it is important to involve the quantized transformer in the prediction stage if the goal is to ultimately separate it from BitTransGNN for solitary use.

## 6.2 The Synergy Between BitTransformer and GNN Models

The core inspiration of BitTransGNN originated from the hypothesis that transformer models and GNNs capture different characteristics of the input data: transformers model local contextual semantics, while GNNs are successful in representing global structural relationships. The synergy between the two is what makes it possible to obtain improvements in performance without substantial increases in cost. To validate this premise, we apply Representational Similarity Analysis (RSA) [108] on the final logits of the BitTransformer,  $\hat{\mathbf{Z}}_{\text{BitTrans}}$ , and the GNN module,  $\mathbf{Z}_{\text{GNN}}$ , which are trained in the BitTransGNN framework.

RSA measures the similarity between representations across models by comparing pairwise distances between data samples of each representation. In our context, we treat the output logits of the BitTransformer and the GNN modules as the model representations. We aim to differentiate the focus of the two models by analyzing how they treat different classes and make class assignments to different samples. In achieving this goal, rather than comparing each output sample, we aim to make a general comment per each class. For that reason, before conducting the RSA, we first take an average over different data instances for each class within the two logits. Then, we create a similarity matrix using the means of each class representation by computing pairwise distances within the two logits. We used the cosine similarity scores between class representations to account for the pairwise distances. We stack the similarity matrices constructed for the BitTransformer and GNN outputs to construct similarity profiles for both methods. We then implement Principal Component Analysis (PCA) on these

similarity profiles to compare the representational focus of the BitTransformer and the GNN models.

We show the first two principal components (PCs) of the outputs of BitTransformer and GNN models that are trained as part of BitTransGNN<sub>Dynamic</sub> on the 20NG dataset for different quantization rates in Fig. 6.2. The PCs of the BitTransformer and GNN output representations are clearly segregated into two clusters in all model variants. These clusters are easily separable, as they are situated far apart along the axis corresponding to the first PC. The similarity profiles of the BitTransformer and GNN output representations can be easily separated by a vertical line. This shows that the BitTransformer and the GNN modules focus on highly distinctive features when constructing their outputs. The difference between the focus of the two models provides a valuable insight into the success of the BitTransGNN framework, and how the GNN module compensates for the shortcomings of transformer models to improve quantized transformer training.

### 6.3 Choice of Quantization Configuration

We investigated different quantization scenarios over BERT components in Table 6.1 to justify our choice of quantization configuration. The efficiency of these scenarios can be evaluated based on the information listed in Tables 4.3 and 4.7. As discussed in Chapter 3.1.1, we quantize linear layer weights to low-bit precision and quantize activations to 8-bits, while maintaining full-precision embeddings in our main model. We refer to this configuration as W1E32A8, where A8 refers to the 8-bit quantization of the activations. This configuration allows for a significant reduction in memory and energy consumption, while retaining a sensible amount of the full-precision model’s expressive power.

Quantizing the activations to 8-bits leads to a slight drop in classification accuracy, generally less than 1%. Although this drop can be seen significant in various applications, utilizing 8-bit activations instead of 32-bit representations allows one to potentially use more efficient data types, such as int32 or int8, which can create

Table 6.1: Study to examine the effect of quantizing different parts of the BERT model on model performance.  $W\{b_l\}E\{b_e\}A\{b_a\}$  refers to a quantization instance of the transformer variant where the embedding weights are quantized to  $b_e$  bits, the parameters in the rest of the modules are quantized to  $b_l$  bits, and the activations right before these parameters are quantized to  $b_a$  bits. BitTransformer models fine-tuned with QAFT are considered.

| BitTransformer | MR               |                  | R8              |                 | R52              |                 | Ohsumed          |                 |
|----------------|------------------|------------------|-----------------|-----------------|------------------|-----------------|------------------|-----------------|
|                | w-F1             | Accuracy         | w-F1            | Accuracy        | w-F1             | Accuracy        | w-F1             | Accuracy        |
| <b>BERT</b>    |                  |                  |                 |                 |                  |                 |                  |                 |
| W1E32A32       | 76.19 $\pm$ 0.8  | 76.25 $\pm$ 0.8  | 96.96 $\pm$ 0.8 | 96.98 $\pm$ 0.8 | 94.58 $\pm$ 0.3  | 94.81 $\pm$ 0.3 | 62.97 $\pm$ 1.2  | 62.96 $\pm$ 1.1 |
| W1E32A8        | 76.82 $\pm$ 0.6  | 76.91 $\pm$ 0.6  | 97.29 $\pm$ 0.5 | 97.30 $\pm$ 0.5 | 94.14 $\pm$ 0.6  | 94.42 $\pm$ 0.7 | 63.74 $\pm$ 0.8  | 63.34 $\pm$ 1.0 |
| W1E32A1        | 56.96 $\pm$ 14.2 | 59.52 $\pm$ 12.1 | 92.97 $\pm$ 3.1 | 93.39 $\pm$ 2.7 | 67.26 $\pm$ 10.8 | 73.50 $\pm$ 7.4 | 15.47 $\pm$ 10.2 | 25.92 $\pm$ 9.2 |
| W1E1A32        | 75.71 $\pm$ 0.8  | 75.90 $\pm$ 0.7  | 97.48 $\pm$ 0.4 | 97.49 $\pm$ 0.4 | 94.02 $\pm$ 0.7  | 94.27 $\pm$ 0.7 | 61.65 $\pm$ 1.6  | 61.87 $\pm$ 0.8 |
| W1E1A8         | 76.34 $\pm$ 0.8  | 76.44 $\pm$ 0.8  | 97.32 $\pm$ 0.5 | 97.34 $\pm$ 0.4 | 94.35 $\pm$ 0.2  | 94.69 $\pm$ 0.3 | 60.96 $\pm$ 1.1  | 61.27 $\pm$ 1.2 |
| W1E1A1         | 61.60 $\pm$ 11.8 | 62.8 $\pm$ 11.2  | 91.57 $\pm$ 4.0 | 92.06 $\pm$ 3.6 | 75.90 $\pm$ 6.7  | 78.30 $\pm$ 5.9 | 23.13 $\pm$ 8.5  | 30.95 $\pm$ 3.3 |
| <b>RoBERTa</b> |                  |                  |                 |                 |                  |                 |                  |                 |
| W1E32A32       | 77.49 $\pm$ 0.5  | 77.50 $\pm$ 0.5  | 97.21 $\pm$ 0.3 | 97.22 $\pm$ 0.3 | 94.38 $\pm$ 0.6  | 94.70 $\pm$ 0.5 | 59.47 $\pm$ 2.7  | 60.36 $\pm$ 1.8 |
| W1E32A8        | 77.25 $\pm$ 0.6  | 77.27 $\pm$ 0.6  | 97.26 $\pm$ 0.2 | 97.27 $\pm$ 0.2 | 94.43 $\pm$ 0.4  | 94.77 $\pm$ 0.5 | 61.54 $\pm$ 0.9  | 61.99 $\pm$ 1.0 |
| W1E32A1        | 76.29 $\pm$ 1.2  | 76.31 $\pm$ 1.2  | 95.28 $\pm$ 0.8 | 95.25 $\pm$ 0.7 | 85.89 $\pm$ 0.4  | 87.36 $\pm$ 0.3 | 38.34 $\pm$ 1.6  | 42.53 $\pm$ 1.3 |
| W1E1A32        | 76.00 $\pm$ 0.4  | 76.04 $\pm$ 0.4  | 97.21 $\pm$ 0.2 | 97.21 $\pm$ 0.2 | 93.31 $\pm$ 0.5  | 93.73 $\pm$ 0.5 | 58.45 $\pm$ 1.4  | 59.30 $\pm$ 1.1 |
| W1E1A8         | 76.54 $\pm$ 0.3  | 76.58 $\pm$ 0.3  | 96.46 $\pm$ 0.5 | 96.48 $\pm$ 0.5 | 94.00 $\pm$ 0.6  | 94.25 $\pm$ 0.5 | 58.14 $\pm$ 0.9  | 58.83 $\pm$ 0.9 |
| W1E1A1         | 76.54 $\pm$ 0.8  | 76.59 $\pm$ 0.7  | 95.43 $\pm$ 0.6 | 95.40 $\pm$ 0.6 | 86.38 $\pm$ 0.4  | 87.72 $\pm$ 0.3 | 36.38 $\pm$ 3.1  | 40.05 $\pm$ 2.0 |

a significant advantage in energy efficiency. For that reason, we quantized the activations to 8-bits. We did not utilize the fully binarized W1E1A1 configuration. Fully binarized configuration loses a lot of representational capability and reports a very large variance in performance metrics. Due to the instability in the reported metrics, we preferred to quantize activations to 8-bits. It might be argued that it is possible to attenuate such loss in model performance caused by the use of binary activations through the use of the BitTransGNN architecture. Despite such an opportunity, since we use a full-precision GNN model within BitTransGNN, we believed that using 1-bit activations within the BitTransformer model would create a large disparity between the two models. For these reasons, we do not use binary activations.

Besides quantizing the linear transformation layers and using 8-bit activations in our model, we keep the embedding module full-precision through our quantization procedure. As displayed in Table 4.3, there is additional potential that can be exploited by also quantizing the embedding module in the transformer model. Although carrying such potential, quantization of the embedding module leads to an additional decrease in performance, as shown in Table 6.1; however, the decrease is modest and it can be argued that the BitTransGNN framework

can effectively eliminate such performance loss.

Our choice of maintaining the embedding module full-precision despite the potential advantage in efficiency was mainly motivated by two reasons. First, the look-up table nature of the embedding module is vastly different from the matrix multiplications that are conducted in the linear transformation layers of the transformer. For this reason, embeddings were found to be more sensitive to quantization [32]. Even though it is not as evident for more scalable transformer models such as BERT and RoBERTa, quantizing the embedding module can lead to a major decrease in model performance because of this sensitivity. Due to such a possible decrease in performance, several works that focused on quantizing larger transformer models and LLMs utilized full-precision embeddings while quantizing the rest of the architectural components to low-bit precision. These models prefer to avoid embedding quantization, as it would lead to a larger decrease in performance compared to the other parts of the model. The error induced by the embedding module in the initial stages of the transformer would propagate to the later layers of the network [20, 33, 109, 110]. Our second motivation was induced by the observation that the advantage which can be achieved by quantizing the embedding module decreases as the size of the transformer model increases. We show an example of such a scenario for quantized BERT and RoBERTa models in Table 6.2. Even for such relatively lightweight transformer models, while transitioning from the base-sized BERT model to the large-sized BERT model, the relative advantage of employing 1-bit quantization in the embedding component significantly decreases, as shown for the W32E1 quantization configuration. The W1E32 configuration that we use in BitTransGNN, which employs binary linear layers while maintaining the embedding module full-precision, displays a much more significant reduction in the memory use of large BERT and RoBERTa models compared to their base-sized counterparts. Hence, the relative contribution of quantizing the embedding module decreases vastly for larger transformer models. Since we want our framework to be scalable and adaptable for transformer networks of different sizes, we also took into account the sensitivities of larger transformer networks. For these reasons, we prefer to use a full-precision embedding module in our quantization procedure.

Table 6.2: Memory consumption of BitTransformer model parameters in different binarization scenarios, listed for different BERT and RoBERTa model sizes.  $W\{b_l\}E\{b_e\}$  refers to embedding weights quantized to  $b_e$  bits and rest of the parameters quantized to  $b_l$  bits.

| Quantization Type    | Size (MB) | Size (Ratio)                     |
|----------------------|-----------|----------------------------------|
| <b>BERT Base</b>     |           |                                  |
| W32E32               | 438.0     | $B_{\text{BERT-BASE}}$           |
| W1E32                | 106.2     | $0.2425B_{\text{BERT-BASE}}$     |
| W32E1                | 345.6     | $0.7891B_{\text{BERT-BASE}}$     |
| W1E1                 | 13.8      | $0.0316B_{\text{BERT-BASE}}$     |
| <b>RoBERTa Base</b>  |           |                                  |
| W32E32               | 498.6     | $B_{\text{RoBERTa-BASE}}$        |
| W1E32                | 166.9     | $0.3347B_{\text{RoBERTa-BASE}}$  |
| W32E1                | 347.5     | $0.6969B_{\text{RoBERTa-BASE}}$  |
| W1E1                 | 15.7      | $0.0315B_{\text{RoBERTa-BASE}}$  |
| <b>BERT Large</b>    |           |                                  |
| W32E32               | 1.441K    | $B_{\text{BERT-LARGE}}$          |
| W1E32                | 168.6     | $0.1170B_{\text{BERT-LARGE}}$    |
| W32E1                | 1.318K    | $0.9146B_{\text{BERT-LARGE}}$    |
| W1E1                 | 45.4      | $0.0315B_{\text{BERT-LARGE}}$    |
| <b>RoBERTa Large</b> |           |                                  |
| W32E32               | 1.522K    | $B_{\text{RoBERTa-LARGE}}$       |
| W1E32                | 249.5     | $0.1639B_{\text{RoBERTa-LARGE}}$ |
| W32E1                | 1.321K    | $0.8676B_{\text{RoBERTa-LARGE}}$ |
| W1E1                 | 48.0      | $0.0315B_{\text{RoBERTa-LARGE}}$ |

## 6.4 Limitations and Drawbacks of BitTrans-GNN Models

A possible drawback of our framework comes from the inefficiencies in terms of the energy costs that can arise if the text graph is too large or too dense in structure. The energy costs are dataset-dependent. To outline the worst case, we mainly base our analysis on the 20NG dataset in our study, which has the largest graph structure and is the most costly dataset to process through BitTransGNN methods. Even for the worst case, our methods show major promise in reducing model costs while significantly improving quantized transformer performance with few exceptions.

In addition to the potential drawbacks that can be caused by the graph structure, the use of the GNN model in every batch update during the execution of BitTransGNN<sub>Dynamic</sub> could induce inefficiency. This characteristic of the dynamic training method leads to an increase in energy consumption, which slightly reduces the potential advantages in energy efficiency that can be achieved through quantization for BitTransGNN<sub>Dynamic</sub> variants. More efficient strategies that would utilize the GNN only once through each epoch or in every few batches, rather than after every batch update, can be explored to improve the efficiency of the dynamic training scheme. Despite the disadvantages during training, it should be noted that the BitTransGNN models trained under the dynamic scheme can be employed at the same expense as BitTransGNN<sub>Static</sub> during inference, since it is possible to employ the GNN module only once after the sequence embeddings are initialized by the BitTransformer. Through such a procedure, BitTransGNN<sub>Dynamic</sub> would require an equal number of operations to that of the BitTransGNN<sub>Static</sub> after training. Regardless, apart from the rare exceptions observed for the 20NG dataset, all BitTransGNN methods offer significant efficiency advantages compared to full-precision models.

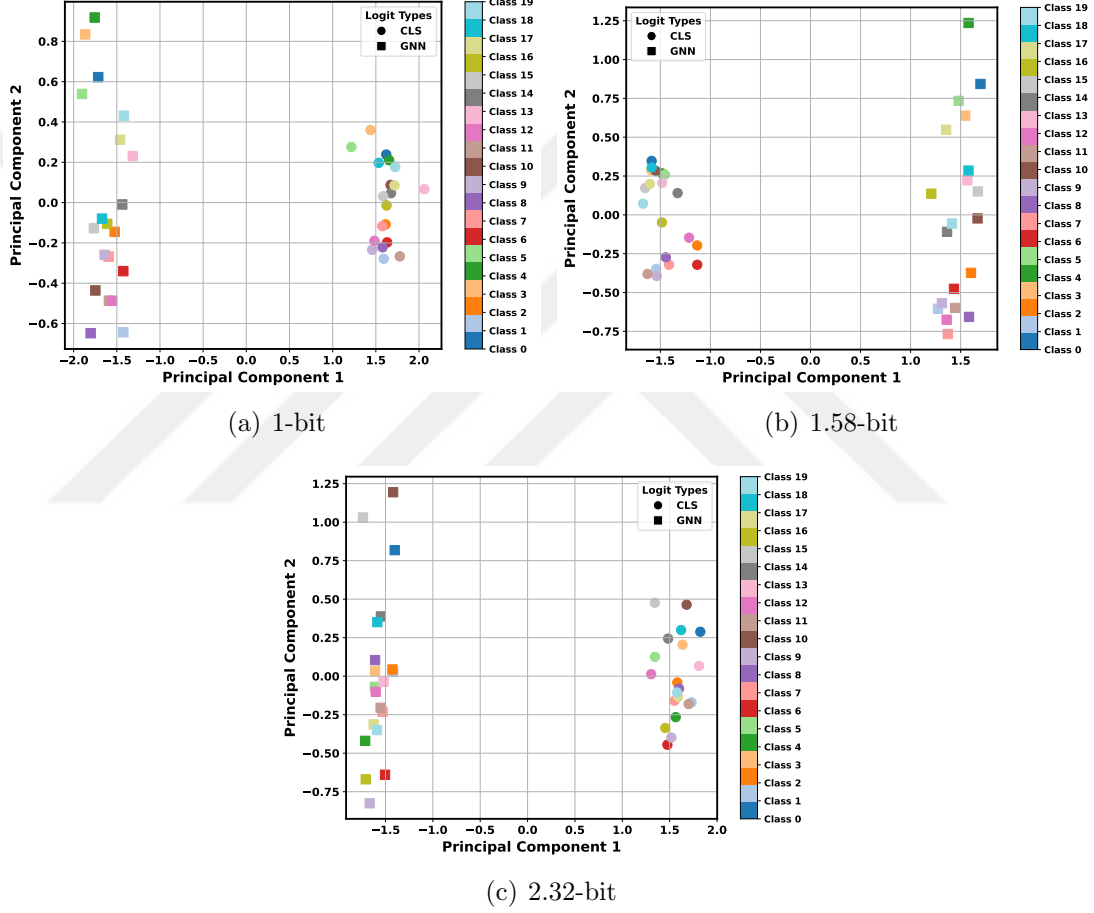


Figure 6.2: PCA analysis on the similarity profiles of the logits of BitTransformer and GNN modules for the BitTransGNN<sub>Dynamic</sub> model trained on the 20NG dataset. Similarity profiles are listed for BitTransGNN<sub>Dynamic</sub> models quantized to different rates. The importance profiles are computed using RSA. Circles correspond to the components from the BitTransformer, and squares correspond to the components from the GNN models of BitTransGNN. Colors represent different classes that the model distinguishes during classification.

## Chapter 7

# Conclusion

In this thesis, we introduced a new approach to improve quantized transformer performance through the use of GNNs. We introduced the BitTransGNN architecture that trains quantized transformer models and GNNs in a joint setting to improve the capabilities of quantized transformers. BitTransGNN assists quantized transformers with information on global structural relationships across data entities to enhance their knowledge representation and improve their training. We first introduced the quantization procedure that we used to quantize the transformer model and how we integrated the GNN into the training stage of the quantized transformer. After setting the foundations of BitTransGNN, we introduced two additional methods that extend the scope of BitTransGNN beyond its transductive nature and adapt it to inductive problem settings by incorporating the knowledge learned by BitTransGNN within a solitary quantized transformer model.

We showed the theoretical advantages in memory and energy efficiency that can be achieved by quantizing the parameters of a full-precision transformer model and that the BitTransGNN architecture can leverage these advantages. The joint architecture only introduces a minor additional cost on top of that of the quantized transformer, and the inductive models can be used with zero additional costs compared to quantized transformer baselines in inference time.

BitTransGNN methods effectively reduced and often closed the performance gap between quantized transformers and their full-precision counterparts. With only a minor additional cost, the transductive BitTransGNN methods outperformed quantized transformer baselines in all tasks and achieved a comparable performance to full-precision transformers in most datasets. In addition to the performance of the transductive model, both inductive BitTransGNN methods significantly improved quantized transformer performance in all tasks with zero additional costs during inference time. Inductive BitTransGNN variants also performed comparable to full-precision transformers in several datasets despite the disadvantages introduced by quantization. Apart from its benefits to our research, the distillation framework that we propose as part of the methods that adapt BitTransGNN to inductive problem settings can be generalized to transfer knowledge across inductive and transductive domains in different studies depending on task needs.

Our study offers a new perspective on improving quantized transformer performance by assisting them with global contextual information captured by GNNs. Our framework can be extended to other domains, such as vision and time-series modeling. It can also be utilized on LLM products to improve their inference efficiency while maintaining a high level of performance.

We believe that the approach we introduce in this thesis will be valuable in reducing the computational overhead of large neural models. Future research in quantization can allow to realize in practice the potential advantages offered by quantization in terms of memory and energy efficiency, opening an important pathway for more efficient and environment-friendly machine learning solutions. Our proposed method will be an important tool to benefit from the advantages of quantization without having to make huge sacrifices from the model capabilities.

# Bibliography

- [1] C. E. Kumbasar, H. M. Ozaktas, and A. Koc, “Graph neural networks improve quantized transformers by incorporating global structural relationships,” *IEEE Transactions on Signal and Information Processing over Networks*, 2025. Manuscript under review.
- [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.
- [3] J. D. M.-W. C. Kenton and L. K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of naacL-HLT*, vol. 1, p. 2, 2019.
- [4] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “RoBERTa: A robustly optimized bert pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.
- [5] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The long-document transformer,” *arXiv preprint arXiv:2004.05150*, 2020.
- [6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” *ICLR*, 2021.

- [7] B. Jiang, K. Zhao, and J. Tang, “RGTransformer: Region-graph transformer for image representation and few-shot classification,” *IEEE Signal Processing Letters*, vol. 29, pp. 792–796, 2022.
- [8] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, “ViViT: A video vision transformer,” in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6836–6846, 2021.
- [9] B. De Weerd, Y. C. Eldar, and N. Deligiannis, “Deep unfolding transformers for sparse recovery of video,” *IEEE Transactions on Signal Processing*, vol. 72, pp. 1782–1796, 2024.
- [10] A. Akman, Q. Sun, and B. W. Schuller, “Improving audio explanations using audio language models,” *IEEE Signal Processing Letters*, pp. 1–5, 2025.
- [11] N. Chen, S. Watanabe, J. Villalba, P. Želasko, and N. Dehak, “Non-autoregressive transformer for speech recognition,” *IEEE Signal Processing Letters*, vol. 28, pp. 121–125, 2021.
- [12] A. Y. Yildiz, E. Koc, and A. Koc, “Multivariate time series imputation with transformers,” *IEEE Signal Processing Letters*, vol. 29, pp. 2517–2521, 2022.
- [13] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, “Scaling laws for neural language models,” *arXiv e-prints*, pp. arXiv–2001, 2020.
- [14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, “Language models are unsupervised multitask learners,”
- [15] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the dangers of stochastic parrots: Can language models be too big?,” in *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pp. 610–623, 2021.

- [16] M. C. Rillig, M. Ågerstrand, M. Bi, K. A. Gould, and U. Sauerland, “Risks and benefits of large language models for the environment,” *Environmental Science & Technology*, vol. 57, no. 9, pp. 3464–3466, 2023.
- [17] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [18] A. Faiz, S. Kaneda, R. Wang, R. C. Osi, P. Sharma, F. Chen, and L. Jiang, “LLMCarbon: Modeling the end-to-end carbon footprint of large language models,” in *The Twelfth International Conference on Learning Representations*.
- [19] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019.
- [20] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, “BitNet: Scaling 1-bit transformers for large language models,” *arXiv e-prints*, pp. arXiv-2310, 2023.
- [21] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.
- [22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *Journal of Machine Learning Research*, vol. 18, no. 187, pp. 1–30, 2018.
- [23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.

- [24] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, “Ternary neural networks for resource-efficient ai applications,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2547–2554, 2017.
- [25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: Imagenet classification using binary convolutional neural networks,” in *Computer Vision – ECCV 2016*, pp. 525–542, Springer International Publishing, 2016.
- [26] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, “Deepshift: Towards multiplication-less neural networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2359–2368, 2021.
- [27] J. Tian, C. Fang, H. Wang, and Z. Wang, “BEBERT: Efficient and robust binary ensemble BERT,” in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, 2023.
- [28] H. Qin, Y. Ding, M. Zhang, Q. YAN, A. Liu, Q. Dang, Z. Liu, and X. Liu, “BiBERT: Accurate fully binarized BERT,” in *International Conference on Learning Representations*, 2022.
- [29] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, “BinaryBERT: Pushing the limit of BERT quantization,” 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)*, pp. 4334–4348, 2021.
- [30] Y. He, Z. Lou, L. Zhang, J. Liu, W. Wu, H. Zhou, and B. Zhuang, “BiViT: Extremely compressed binary vision transformers,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5651–5663, October 2023.
- [31] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, “Q8BERT: Quantized 8bit BERT,” in *2019 Fifth Workshop on Energy Efficient Machine Learning*

*and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pp. 36–39, 2019.

- [32] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, “Q-BERT: Hessian based ultra low precision quantization of bert,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 8815–8821, 2020.
- [33] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, “The era of 1-bit LLMs: All large language models are in 1.58 bits,” *arXiv e-prints*, pp. arXiv–2402, 2024.
- [34] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *International Conference on Learning Representations*, 2017.
- [35] M. Henaff, J. Bruna, and Y. LeCun, “Deep convolutional networks on graph-structured data,” *arXiv e-prints*, pp. arXiv–1506, 2015.
- [36] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and deep locally connected networks on graphs,” in *2nd International Conference on Learning Representations, ICLR 2014*, 2014.
- [37] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272, PMLR, 06–11 Aug 2017.
- [38] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” *International Conference on Learning Representations*, 2018.
- [39] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” *Advances in neural information processing systems*, vol. 30, 2017.

- [40] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, “Simplifying graph convolutional networks,” in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871, PMLR, 09–15 Jun 2019.
- [41] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, “CayleyNets: Graph convolutional neural networks with complex rational spectral filters,” *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2019.
- [42] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” *Advances in neural information processing systems*, vol. 29, 2016.
- [43] V. N. Vapnik, “Statistical learning theory,” *John Wiley & Sons*, vol. 2, 1998.
- [44] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf, “Learning with local and global consistency,” *Advances in neural information processing systems*, vol. 16, 2003.
- [45] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *NIPS Deep Learning and Representation Learning Workshop*, 2015.
- [46] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.
- [47] V. Sanh, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” *arXiv preprint arXiv:1910.01108*, 2019.
- [48] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, “MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 5776–5788, 2020.
- [49] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, “TinyBERT: Distilling BERT for natural language understanding,”

- in *Findings of the Association for Computational Linguistics: EMNLP 2020* (T. Cohn, Y. He, and Y. Liu, eds.), pp. 4163–4174, Association for Computational Linguistics, Nov. 2020.
- [50] Z. Liu, B. Oguz, A. Pappu, L. Xiao, S. Yih, M. Li, R. Krishnamoorthi, and Y. Mehdad, “BiT: Robustly binarized multi-distilled transformer,” *Advances in neural information processing systems*, vol. 35, pp. 14303–14316, 2022.
  - [51] P. Chen, B. Zhuang, and C. Shen, “FATNN: Fast and accurate ternary neural networks,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5219–5228, October 2021.
  - [52] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, “Estimating the carbon footprint of BLOOM, a 176b parameter language model,” *Journal of Machine Learning Research*, vol. 24, no. 253, pp. 1–15, 2023.
  - [53] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains,” *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.
  - [54] A. Sandryhaila and J. M. Moura, “Discrete signal processing on graphs,” *IEEE transactions on signal processing*, vol. 61, no. 7, pp. 1644–1656, 2013.
  - [55] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, “Graph signal processing: Overview, challenges, and applications,” *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.
  - [56] A. Sandryhaila and J. M. Moura, “Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure,” *IEEE signal processing magazine*, vol. 31, no. 5, pp. 80–90, 2014.
  - [57] A. Ortega, *Introduction to graph signal processing*. Cambridge University Press, 2022.

- [58] A. Sandryhaila and J. M. Moura, “Discrete signal processing on graphs: Frequency analysis,” *IEEE Transactions on signal processing*, vol. 62, no. 12, pp. 3042–3054, 2014.
- [59] A. Gavili and X.-P. Zhang, “On the shift operator, graph frequency, and optimal filtering in graph signal processing,” *IEEE Transactions on Signal Processing*, vol. 65, no. 23, pp. 6303–6318, 2017.
- [60] J. Shi and J. M. Moura, “Graph signal processing: Dualizing gsp sampling in the vertex and spectral domains,” *IEEE Transactions on Signal Processing*, vol. 70, pp. 2883–2898, 2022.
- [61] A. G. Marques, S. Segarra, G. Leus, and A. Ribeiro, “Stationary graph processes and spectral estimation,” *IEEE Transactions on Signal Processing*, vol. 65, no. 22, pp. 5911–5926, 2017.
- [62] T. Alikasifoglu, B. Kartal, E. Ozgunay, and A. Koc, “Joint time-vertex fractional Fourier transform,” *Signal Processing*, vol. 233, p. 109944, 2025.
- [63] T. Alikasifoglu, B. Kartal, and A. Koc, “Graph fractional Fourier transform: A unified theory,” *IEEE Transactions on Signal Processing*, 2024.
- [64] T. Alikasifoglu, B. Kartal, and A. Koc, “Wiener filtering in joint time-vertex fractional Fourier domains,” *IEEE Signal Processing Letters*, vol. 31, pp. 1319–1323, 2024.
- [65] B. Kartal, Y. E. Bayiz, and A. Koc, “Graph signal processing: Vertex multiplication,” *IEEE Signal Processing Letters*, vol. 28, pp. 1270–1274, 2021.
- [66] C. Ozturk, H. M. Ozaktas, S. Gezici, and A. Koc, “Optimal fractional fourier filtering for graph signals,” *IEEE Transactions on Signal Processing*, vol. 69, pp. 2902–2912, 2021.
- [67] E. Bayram, P. Frossard, E. Vural, and A. Alatan, “Analysis of airborne LiDAR point clouds with spectral graph filtering,” *IEEE geoscience and remote sensing letters*, vol. 15, no. 8, pp. 1284–1288, 2018.

- [68] S. Kwak, N. Geroliminis, and P. Frossard, “Traffic signal prediction on transportation networks using spatio-temporal correlations on graphs,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 7, pp. 648–659, 2021.
- [69] R. A. Sevimli, M. Ucuncu, and A. Koc, “RadGT: Graph and Transformer-Based Automotive Radar Point Cloud Segmentation,” *IEEE Sensors Letters*, vol. 7, no. 11, pp. 1–4, 2023.
- [70] R. A. Sevimli, M. Ucuncu, and A. Koc, “Graph signal processing based object classification for automotive radar point clouds,” *Digital Signal Processing*, vol. 137, p. 104045, 2023.
- [71] H.-T. Wai, Y. C. Eldar, A. E. Ozdaglar, and A. Scaglione, “Community inference from partially observed graph signals: Algorithms and analysis,” *IEEE Transactions on Signal Processing*, vol. 70, pp. 2136–2151, 2022.
- [72] J. Yang, S. C. Draper, and R. Nowak, “Learning the interference graph of a wireless network,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 3, no. 3, pp. 631–646, 2016.
- [73] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [74] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mc-clelland. vol. 1. 1986,” *Biometrika*, vol. 71, no. 599-607, p. 6, 1986.
- [75] S. Hochreiter, “Long short-term memory,” *Neural Computation MIT-Press*, 1997.
- [76] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*

- (*EMNLP*), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.
- [77] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” in *International conference on machine learning*, pp. 933–941, PMLR, 2017.
  - [78] A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” *arXiv preprint arXiv:2312.00752*, 2023.
  - [79] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *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*, pp. 38–45, 2020.
  - [80] H. Zhang and J. Zhang, “Text graph transformer for document classification,” in *Conference on empirical methods in natural language processing (EMNLP)*, 2020.
  - [81] F. Sahinuc and A. Koc, “Fractional fourier transform meets transformer encoder,” *IEEE Signal Processing Letters*, vol. 29, pp. 2258–2262, 2022.
  - [82] L. Wu, Y. Chen, K. Shen, X. Guo, H. Gao, S. Li, J. Pei, B. Long, *et al.*, “Graph neural networks for natural language processing: A survey,” *Foundations and Trends® in Machine Learning*, vol. 16, no. 2, pp. 119–328, 2023.
  - [83] L. Yao, C. Mao, and Y. Luo, “Graph convolutional networks for text classification,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 7370–7377, Jul. 2019.
  - [84] R. Ragesh, S. Sellamanickam, A. Iyer, R. Bairi, and V. Lingam, “HeteGCN: Heterogeneous graph convolutional networks for text classification,” in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining, WSDM ’21*, p. 860–868, Association for Computing Machinery, 2021.

- [85] Y. Lin, Y. Meng, X. Sun, Q. Han, K. Kuang, J. Li, and F. Wu, “BertGCN: Transductive text classification by combining GNN and BERT,” in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 1456–1462, 2021.
- [86] A. C. Aras, T. Alikasifoglu, and A. Koc, “Graph receptive transformer encoder for text classification,” *IEEE Transactions on Signal and Information Processing over Networks*, vol. 10, pp. 347–359, 2024.
- [87] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima’an, “Graph convolutional encoders for syntax-aware neural machine translation,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (M. Palmer, R. Hwa, and S. Riedel, eds.), (Copenhagen, Denmark), pp. 1957–1967, Association for Computational Linguistics, Sept. 2017.
- [88] D. Beck, G. Haffari, and T. Cohn, “Graph-to-sequence learning using gated graph neural networks,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (I. Gurevych and Y. Miyao, eds.), (Melbourne, Australia), pp. 273–283, Association for Computational Linguistics, July 2018.
- [89] D. Wang, P. Liu, Y. Zheng, X. Qiu, and X. Huang, “Heterogeneous graph neural networks for extractive document summarization,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, eds.), pp. 6209–6219, Association for Computational Linguistics, July 2020.
- [90] P. Fernandes, M. Allamanis, and M. Brockschmidt, “Structured neural summarization,” in *International Conference on Learning Representations*, 2019.
- [91] A. C. Aras, T. Alikasifoglu, and A. Koc, “Text-RGNNs: Relational Modeling for Heterogeneous Text Graphs,” *IEEE Signal Processing Letters*, vol. 31, pp. 1955–1959, 2024.

- [92] Z. Chen, H. Mao, H. Li, W. Jin, H. Wen, X. Wei, S. Wang, D. Yin, W. Fan, H. Liu, *et al.*, “Exploring the potential of large language models (llms) in learning on graphs,” *ACM SIGKDD Explorations Newsletter*, vol. 25, no. 2, pp. 42–61, 2024.
- [93] C. Qian, H. Tang, Z. Yang, H. Liang, and Y. Liu, “Can large language models empower molecular property prediction?,” *arXiv preprint arXiv:2307.07443*, 2023.
- [94] J. Tang, Y. Yang, W. Wei, L. Shi, L. Su, S. Cheng, D. Yin, and C. Huang, “GraphGPT: Graph instruction tuning for large language models,” in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 491–500, 2024.
- [95] T. Alikasıfoğlu, A. Aras, and A. Koc, “VISPool: Enhancing transformer encoders with vector visibility graph neural networks,” in *Findings of the Association for Computational Linguistics: ACL 2024* (L.-W. Ku, A. Martins, and V. Srikumar, eds.), (Bangkok, Thailand), pp. 2547–2556, Association for Computational Linguistics, Aug. 2024.
- [96] E. Simsek, A. Topcu, E. Koc, E. U. Saritas, and A. Koc, “Emotion classification with visibility graphs,” *IEEE Signal Processing Letters*, vol. 32, pp. 2055–2059, 2025.
- [97] E. Koc, E. Kulkul, G. Kaynar, M. Acar, T. Cukur, and A. Koc, “scGraPhT: Merging Transformers and Graph Neural Networks for Single-Cell Annotation,” *IEEE Transactions on Signal and Information Processing over Networks*, 2025. in press.
- [98] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.
- [99] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv preprint arXiv:1308.3432*, 2013.

- [100] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, “Learning efficient object detection models with knowledge distillation,” *Advances in neural information processing systems*, vol. 30, 2017.
- [101] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “FitNets: Hints for thin deep nets,” in *In Proceedings of ICLR*, 2015.
- [102] S. Zagoruyko and N. Komodakis, “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer,” *arXiv preprint arXiv:1612.03928*, 2016.
- [103] J. Kim, S. Park, and N. Kwak, “Paraphrasing complex network: Network compression via factor transfer,” *Advances in neural information processing systems*, vol. 31, 2018.
- [104] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014.
- [105] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, “Ten lessons from three generations shaped google’s tpuv4i : Industrial product,” in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, 2021.
- [106] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, “GLUE: A multi-task benchmark and analysis platform for natural language understanding,” in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (T. Linzen, G. Chrupala, and A. Alishahi, eds.), (Brussels, Belgium), pp. 353–355, Association for Computational Linguistics, Nov. 2018.
- [107] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimeshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner,

- L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.
- [108] N. Kriegeskorte, M. Mur, and P. A. Bandettini, “Representational similarity analysis-connecting the branches of systems neuroscience,” *Frontiers in systems neuroscience*, vol. 2, p. 249, 2008.
- [109] 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*, pp. 38087–38099, PMLR, 2023.
- [110] E. Frantar, S. Ashkboos, T. Hoefer, and D. Alistarh, “Gptq: Accurate post-training quantization for generative pre-trained transformers,” in *The Eleventh International Conference on Learning Representations*, OpenReview, 2023.

# Appendix A

## Source Code

The codebase of the thesis and the datasets used in the experiments, as well as notebook files that outline the calculations relevant to the efficiency chapters, can be found in the GitHub repository located in the link <https://github.com/koc-lab/BitTransGNN>: