# Graph Neural Networks Improve Quantized Transformers by Incorporating Global Relationships

Cihan Eralp Kumbasar, Haldun M. Ozaktas, *Fellow, IEEE*, and Aykut Koç, *Senior Member, IEEE*

***Abstract*—Transformers are established in natural language processing (NLP). Their capabilities improve with model size, but training and hosting larger models are computationally demanding. Quantization of model parameters can reduce cost; however, it generally leads to a significant loss in model performance. We introduce BitTransGNN which improves the performance of transformers quantized to low-bit precision by integrating their architecture with Graph Neural Networks (GNNs) during training. BitTransGNN substantially reduces and often closes the performance gap between quantized transformers and their full-precision counterparts while retaining the computational efficiency advantages provided by quantization, including lower memory and energy costs. This is possible since transformers and GNNs capture different characteristics of data. Transformers excel in modeling contextual semantics. GNNs excel in modeling very long-range relationships extracted from the entirety of texts beyond the reach of transformers' contextual window sizes. Alongside the transductive joint model, we introduce inductive variants of BitTransGNN, which transfer the knowledge of the joint framework into a standalone quantized transformer. BitTransGNN outperforms quantized transformer baselines trained without GNN integration with minimal additional overhead, and shows competitive performance against full-precision baselines across multiple language benchmarks. Its inductive variants also successfully encapsulate the information captured by the joint model into a standalone quantized transformer, achieving consistent performance gains over quantized baselines with zero additional inference cost. Furthermore, by combining advanced post-training quantization methods with BitTransGNN, we show that its improvements are complementary to those attainable through advancements in quantization.***

***Index Terms*—Transformers, quantization, binary neural networks, efficient transformers, large language models (LLMs), graph neural network (GNN), graph convolutional network (GCN), knowledge distillation, BERT.**

Received 13 February 2025; revised 14 October 2025 and 2 December 2025; accepted 10 January 2026. Date of publication 16 January 2026; date of current version 2 February 2026. The work of Haldun M. Ozaktas was supported in part by the Turkish Academy of Sciences. The work of Aykut Koç work was supported by the BAGEP Award of the Science Academy. This work was supported by TUBITAK 2210-A National Graduate Student Scholarship Program and TUBITAK 1001 under Grant 124E179. The associate editor coordinating the review of this article and approving it for publication was Dr. Nicolas Keriven. *(Corresponding author: Aykut Koç.)*

Cihan Eralp Kumbasar and Aykut Koç are with the Department of Electrical and Electronics Engineering and UMRAM, Bilkent University, TR-06800 Ankara, Türkiye (e-mail: aykut.koc@bilkent.edu.tr).

Haldun M. Ozaktas is with the Department of Electrical and Electronics Engineering, Bilkent University, TR-06800 Ankara, Türkiye (e-mail: haldun@ee.bilkent.edu.tr).

This article has supplementary downloadable material available at https://doi.org/10.1109/TSIPN.2026.3654839, provided by the authors.

Digital Object Identifier 10.1109/TSIPN.2026.3654839

## I. Introduction

TRANSFORMERS achieve exceptional success in almost all natural language processing (NLP) tasks, such as machine translation, named entity recognition, and language modeling [1], [2], [3], [4]. The impressive representational capabilities of transformers have also received interest in other domains such as image, video, audio, and speech processing [5], [6], [7], [8], [9], [10].

The performance of neural models improves with increasing model size [11], which has led to the use of larger transformers in language processing tasks. This eventually led to the introduction of Large Language Models (LLMs) that contain highly parameterized transformer networks at their cores [12]. While displaying unmatched performance, training and hosting larger models is challenging due to increased computational costs [13], limiting the accessibility of these models. Beyond the financial cost, the use of computationally demanding models increases energy consumption and carbon footprint, which raises environmental, ethical, and regulatory concerns [14]. As a striking example, training the Generative Pre-Trained Transformer 3 (GPT-3) [15] model containing 175 billion parameters has a carbon footprint of over 552,100 kilograms of $CO_2$-equivalents [16], corresponding to more than 110 times the average yearly $CO_2$ emission by a human being [17]. Interest in hosting transformer-based models on mobile devices provides further incentive to reduce computational cost [18].

Quantization maps high-precision data to a smaller set of discrete values, thus compressing full-precision parameters into lower-bit representations [19]. It has been instrumental in improving the computational efficiency of early deep learning models [20], [21], [22], [23], [24], [25]. The limiting case of quantization is binarization, which constrains model weights to the set $\{-1,1\}$. Such mapping can reduce memory costs by storing model parameters with single bits, and decrease computational costs by replacing resource-intensive matrix multiplication operations with additions and subtractions. Due to its strong potential in reducing computational costs and memory overhead, quantization has become a key tool for improving transformer inference efficiency. Quantized versions of different transformer models have been introduced to reduce the computation and memory overhead of these models [26], [27], [28], [29], [30], [31].

Despite its promise to improve computational efficiency, binarization leads to a significant drop in transformer model performance [27], [28]. Some methods use relatively higher precision during quantization to reduce this performance loss (e.g., 8

2373-776X © 2026 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

bits or 16 bits) [30] while making significant sacrifices from the potential advantages in efficiency. 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 [18]. Training with the binary parameter set produces better results [18], [32]; however, the methods still underperform compared to their full-precision counterparts due to reduced representational capabilities.

In this paper, we introduce BitTransGNN, a novel framework that leverages Graph Neural Networks (GNNs) [33], [34], [35], [36], [37], [38], [39], [40] to improve quantized transformer training. BitTransGNN reduces the performance gap between quantized and full-precision transformers with minimal additional overhead by supplying very long-range relational information extracted from the entirety of texts beyond the reach of transformers' contextual window sizes.

We now discuss the effect of transformer window size. Transformers' profound success is mainly built on their ability to grasp contextual information within sequences through the use of a finite-context self-attention window [1]. Despite its successful representation of this local context, self-attention is incapable of processing any information contained outside its window. Increasing the span of the window is costly, 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 that are beyond the reach of attention windows. For the purpose of this paper, we will refer to information relationships within the window as local and those beyond it as global. Architectures such as gated linear units [41], recurrent models [42], [43], [44], [45], and state-space models [46] capture long-range dependencies with sub-quadratic time-complexity but have difficulty focusing on salient input regions, leading to inferior performance compared to transformers [47]. Similarly, low-rank attention variants achieve sub-quadratic complexity at the cost of reduced performance [4], [48], [49], [50].

Motivated by these limitations, we aim to improve the knowledge representation of quantized transformers by compensating for the inherent limitation of transformers in modeling long-range dependencies. Several studies have explored graph representations as an alternative to sequential modeling to capture relationships between distant entities [51], [52], [53], [54], [55]. GNNs naturally exploit such distant relationships by propagating information over graph edges to model global dependencies. By modeling global context, GNNs complement transformer models, which excel at learning local contextual patterns. We leverage this complementary nature to mitigate the performance loss from quantization by supplying quantized transformers with global information captured by GNNs. The synergy between the two allows us to overcome the trade-off between cost and accuracy that comes with quantization.

To this end, we propose a GNN-based framework that models relationships among data entities as a graph and updates node representations by aggregating neighborhood information through graph convolutions [33], [35], [39], [40], [56]. Graph Convolutional Networks (GCNs) [33] exhibit linear complexity with respect to the number of graph edges and leverage graph sparsity to reduce computations, enabling them to operate with significantly less energy than transformers. GNNs also train far fewer parameters compared to transformers, making them more memory efficient. We enhance the performance of quantized transformers by utilizing these advantages of GNNs during training. Having established the foundations of BitTransGNN, we extend its applicability to both transductive and inductive learning settings [57]. In transductive learning, the model leverages the structure of the entire dataset to make inferences on specific test cases [58], whereas inductive learning infers generalizable rules to handle unseen inputs [38]. BitTransGNN is inherently transductive due to the direct involvement of the graph convolutional model in its final predictions. To extend its use to inductive tasks, we propose two methods that encapsulate the information captured by the joint BitTransGNN model within an independent quantized transformer. The first directly separates the quantized transformer from the joint model for inference without additional training. The second method adopts a lightweight knowledge distillation approach [59], [60], [61], [62], [63], [64], [65] where a trained BitTransGNN acts as the teacher and a quantized transformer as the student model. Through this setup, the student quantized transformer learns to mimic the behavior of the teacher BitTransGNN, effectively distilling the learned representations captured by the joint design into a standalone quantized transformer. Both methods remove the quantized transformer's dependence on the GNN after BitTransGNN training, extend BitTransGNN's utility to inductive tasks, and maintain the efficiency benefits of quantization during inference.

Our contributions are summarized as follows:

- We propose BitTransGNN to substantially reduce, and often close, the performance gap between quantized transformers and their full-precision counterparts while retaining the efficiency advantages provided by quantization.
- We introduce inductive variants of BitTransGNN that encapsulate the knowledge of the joint framework into a standalone quantized transformer, demonstrating notable performance gains with zero additional inference cost.
- We show that BitTransGNN prevents, to a substantial degree, the performance loss that would otherwise accompany a major reduction in memory and energy costs across diverse sequence-level NLP tasks, including classification, entailment, and semantic similarity, while introducing only minor additional overhead.
- We show that the improvements achieved by BitTransGNN are complementary to developments in quantization techniques, and that it can be combined with state-of-the-art quantization methods for additional performance gains.
- We provide insights into the distinct yet synergistic roles of GNNs and transformers through a comparative analysis of their output profiles.

The rest of the manuscript 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 Section II. Then, we present the BitTransGNN in Section III. Section IV presents the experimental setup, model performances under different configurations,

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

hardware measurements, and ablation studies. We provide discussions to assess the significance of our findings in Section V. Finally, we conclude in Section VI.

## II. Related Work

In this section, we first cover research that applies quantization techniques to reduce the memory footprint and computational cost of neural networks and transformers. Next, we examine the use of GNNs in NLP tasks to capture long-range dependencies that traditional sequence-based models have difficulty representing. Finally, we discuss approaches that address the high computational demands of large models by transferring knowledge from larger models to smaller ones through knowledge distillation.

### A. Quantization in Neural Networks

Quantization and binarization strategies have long been explored to improve the efficiency of deep learning models. Prior works such as [22] and [20] train a neural network with binary weights and activations, demonstrating substantial reductions in computational cost by simplifying arithmetic operations. Binarization maps real-valued neural network parameters to the set {-1,1}, allowing the replacement of computationally demanding multiplication operations with additions and subtractions. The ternary version also carries such potential through the set $\{-1, 0, 1\}$ [23], [66]. Alternative sets, such as $\{-2, -1, 0, 1, 2\}$, can also provide such an advantage, as multiplication can also be eliminated in this case by using shift operations in binary arithmetic [24], [25].

Quantization has been extensively used in transformer models to reduce their enormous computational demands. Early approaches focused on reducing 32-bit full-precision weights to 8-bit representations [30] and applying mixed-precision quantization by adaptively assigning bit-widths using Hessian information [31]. Beyond higher bit-width quantization, researchers explored extremely low-bit settings as well. BEBERT [26], BiBERT [27], and BinaryBERT [28] investigated binary BERT variants for language tasks, while BiViT introduced a binary vision transformer [29]. Quantization research was naturally extended to LLMs, as they incur significant computational and environmental costs due to their larger scaling [67]. Specialized kernels for quantized matrix operations were developed to improve inference efficiency and reduce deployment costs of LLM products [68], [69], [70].

Several studies trained binary and ternary LLMs from scratch to exploit the potential efficiency savings during inference [18], [32]. Despite improving performance and promising reductions in inference costs, quantization-aware training methods introduce additional training overhead. Post-training quantization strategies have been proposed to avoid the additional costs. Methods such as Outlier Suppression (OS), OS+, and SpinQuant target eliminating rarely observed, extremely large activation values in transformer layers to improve quantization performance [71], [72], [73]. Other methods adaptively assign bit-widths to transformer parameters or exploit second-order Hessian information to reduce quantization error [74], [75], [76]. Although these approaches eliminate the cost of retraining, they often face difficulties in settings involving extremely low-bit precision. Prior work has also used GNNs to model dependencies among components for parameter bit-width optimization in mixed-precision quantization [77], [78]. These works use graphs to represent dependencies between model parameters to improve the core quantization process. In contrast, our approach uses graphs to capture dependencies between data elements, complementing the sequential modeling of the quantized transformer to enrich its representation capability, rather than directly controlling the quantization process. We focus on global data-level relationships to enhance the representational capacity of quantized transformers, a perspective that has received no attention to date. In other words, our work takes a different perspective. Instead of modifying the quantization process, we address the representational limitations of transformers by injecting very long-range information captured by a GNN into the quantized model.

### B. Graph Neural Networks in Natural Language Processing

Earlier work on NLP utilized recurrent methods, such as Recurrent Neural Network (RNN) [42], [43], Long-Short Term Memory (LSTM) [44], and Gated Recurrent Unit (GRU) [45], as well as methods that employ gated convolutions [41], to capture the local context within sequences. These models were capable of handling long sequences with sub-quadratic time complexity; however, they lacked a selection mechanism that detects important information while handling long-range dependencies [47]. 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 [1]. Modifications to the self-attention mechanism have been investigated to reduce its quadratic complexity [4], [48], [49], [50], with some achieving linear complexity through low-rank approximations [48] or windowed attention patterns [4]. However, these methods usually come at the cost of reduced training stability or model performance [79]. A supporting approach to the sequential processing of text is modeling the entire corpus through graph structures to capture the global context and depict relationships that are beyond the reach of the local context. GNNs appeared as a natural tool to process information through text graphs. Due to the recent success of this approach, GNN-based methods have gained popularity in NLP tasks [80], including text classification [51], [52], [53], [54], [55], neural machine translation [81], [82], and text summarization [83], [84].

TextGCN [51], a pioneering work that utilizes GNNs in NLP tasks, approaches text classification from a graph perspective. The authors place the words and documents within a corpus to the nodes of a text graph and determine the relations between nodes through text statistics. TextGCN uses a homogeneous GNN to traverse through the heterogeneous text graph, updating word and document embeddings identically [51]. In heterogeneous graphs, the relationships between different node types can be partitioned with some additional overhead to improve their representations, and the data can be processed using

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

heterogeneous GNNs [52], [85], [86]. HeteGCN [52] exploits the heterogeneous structure of TextGCN's text graph by proposing different message propagation paths between word and document nodes.

Beyond the direct use of GNNs in NLP tasks, several works combine the strengths of transformers and GNNs. BertGCN [54] jointly trains a fine-tuned BERT model with a GCN, while the Graph Receptive Transformer Encoder (GRTE) [55] integrates BERT-produced document embeddings into graph nodes and propagates information through the heterogeneous paths proposed in HeteGCN. Similar collaborations between GNNs and LLMs have also been explored in diverse settings to improve the performance of both systems [87], [88], [89], [90], [91]. These approaches demonstrate the benefit of introducing global relationships captured by GNNs into transformer architectures; however, none have examined this direction to enhance the representational capacity of quantized transformers.

More recently, GNNs have also been used to support mixed-precision quantization of transformers. For instance, [77] employs a GNN to capture dependencies among transformer parameters for adaptive bit-width assignment, while [78] uses a graph transformer to model component dependencies and optimize quantization. Although these studies exploit GNNs to represent relationships between model components, our work adopts a distinct perspective: we model global relationships across data entities and inject this information into a quantized transformer during training, thus improving its representational capacity under reduced bit-precision.

### C. *Knowledge Distillation Over Transformer Networks*

Building on the idea that the knowledge of multiple models can be compressed into a single one [60], [59] introduced knowledge distillation, a technique that transfers knowledge from a larger, highly parameterized teacher model to a smaller student model. Distillation enhances the student model's performance by training it to align its output distributions with those of the teacher model while simultaneously learning from ground-truth labels.

Many works leveraged distillation to compress transformer models and obtain smaller models with fewer parameters that are almost as capable as their full-precision counterparts. Methods such as DistilBert [61], TinyBert [63], and MiniLM [62] utilize distillation to reduce the model size of BERT-based language models. Inspired by the method's success in model compression, many works leverage distillation for transferring the capabilities of full-precision transformers into quantized models as a means for improving 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 variants [64]. BEBERT [26] integrates multiple binary and full-precision BERT models through ensemble learning and explores several distillation approaches to train these ensembles and improve binary BERT performance. In BiBERT [27], the authors claim that binary BERT training with distillation can suffer due to a direction mismatch during backpropagation. As a remedy, they propose a Direction Matching Distillation scheme to train a binary BERT. Motivated by these studies, we introduce a teacher–student design built on the collaborative transformer–GNN design and use knowledge distillation to transfer information from the joint model into a student quantized transformer, thereby enhancing its representational capacity.

TABLE I
SETS USED TO QUANTIZE THE FULL-PRECISION WEIGHTS

| | 1-bit | 1.58-bit | 2-bit | 2.32-bit |
|---|---|---|---|---|
| $\mathcal{S}$ | $\{-1,1\}$ | $\{-1,0,1\}$ | $\{-2,-1,1,2\}$ | $\{-2,-1,0,1,2\}$ |

## III. BITTRANSGNN

A visual representation of our proposed BitTransGNN architecture is presented in Fig. 1. We mainly operate BitTransGNN on sequence-level tasks, which correspond to document-level tasks in the NLP context. We will use the two terms interchangeably. We refer to transformers quantized to low-bit precision as BitTransformers in our study and as BitTrans in equations for brevity. We first explain the transformer quantization procedure and the information processing of quantized transformer blocks in Section III-A. We outline the text-graph generation and how the GNN module processes the graph-structured data in Section III-B. We formulate how BitTransGNN is built upon the BitTransformer and GNN modules in Section III-C. We introduce the methods used to train BitTransGNN in Section III-D. We present two extensions that adapt the framework to inductive problem settings by encapsulating the knowledge learned by BitTransGNN models into a quantized transformer in Section III-E. Finally, we theoretically analyze the memory and energy efficiency of BitTransGNN models in Section III-F.

### A. *BitTransformer Module*

We first outline the procedure used to quantize the transformer model. Then, we formulate how the quantized transformer processes information across its blocks within the BitTransGNN framework.

*1) Quantization Procedure:* We quantize the real-valued weights within the transformer network to low-bit precision following the approaches in [92]. Each value in the weight matrices is mapped 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 I.

The conventional matrix multiplication in a linear transformation layer is:

$$\mathbf{Y}_{\text{lin}} = \mathbf{X}_{\text{lin}}\mathbf{W}_{\text{lin}}, \tag{1}$$

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:

$$\widetilde{\mathbf{W}}_{\text{lin}} = \text{Clamp}\left(\text{round}\left(\frac{\mathbf{W}_{\text{lin}}}{\alpha_{\text{W}}}\right), \min(\mathcal{S}), \max(\mathcal{S})\right), \tag{2}$$

$$\text{Clamp}(x, a, c) = \min\left(\max(x, a), c\right), \quad x, a, c \in \mathbb{R}. \tag{3}$$

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

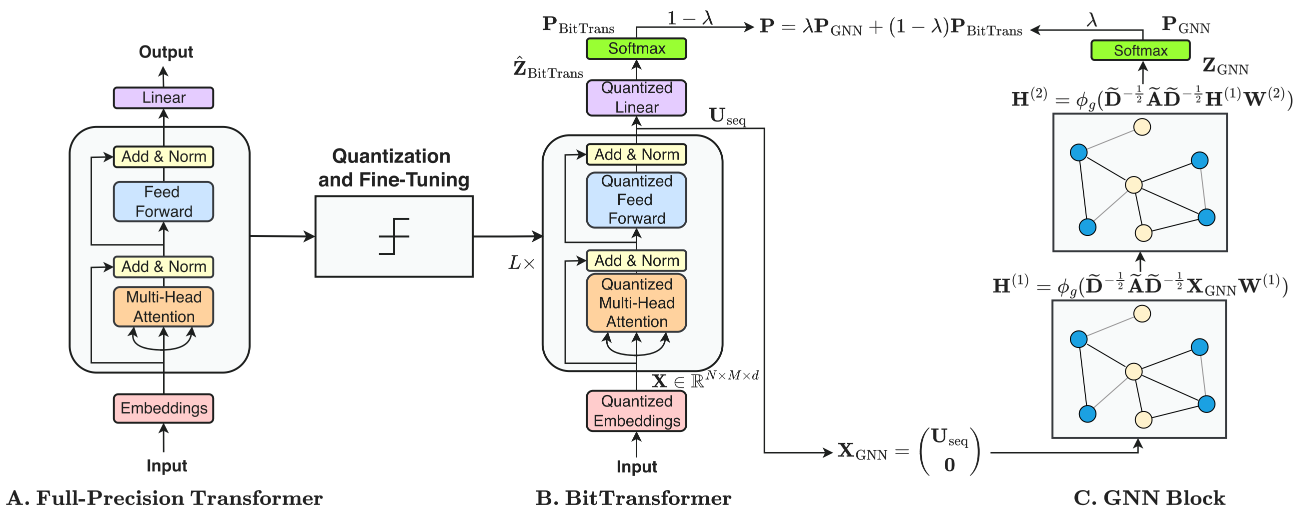
Output
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Embeddings
Input
A. Full-Precision Transformer
Quantization and Fine-Tuning
$L\times$
$\mathbf{P}_{\text{BitTrans}}$
$1-\lambda$
$\mathbf{P} = \lambda \mathbf{P}_{\text{GNN}} + (1-\lambda)\mathbf{P}_{\text{BitTrans}}$
$\lambda$
$\mathbf{P}_{\text{GNN}}$
Softmax
$\hat{\mathbf{Z}}_{\text{BitTrans}}$
Quantized Linear
$\mathbf{U}_{\text{seq}}$
Quantized Feed Forward
Quantized Multi-Head Attention
$\mathbf{X} \in \mathbb{R}^{N\times M\times d}$
Quantized Embeddings
B. BitTransformer
$\mathbf{Z}_{\text{GNN}}$
$\mathbf{H}^{(2)} = \phi_g(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(1)}\mathbf{W}^{(2)})$
$\mathbf{H}^{(1)} = \phi_g(\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}_{\text{GNN}}\mathbf{W}^{(1)})$
$\mathbf{X}_{\text{GNN}} = \begin{pmatrix}\mathbf{U}_{\text{seq}}\\ \mathbf{0}\end{pmatrix}$
C. GNN Block

Fig. 1. Overview of the proposed BitTransGNN architecture. The figure illustrates the transformer quantization and fine-tuning, the GNN block used alongside the BitTransformer, and their integration for joint training. The depicted BitTransGNN employs an L-layer BitTransformer and a 2-layer GCN block.

The function $\mathrm{round}(x) : \mathbb{R} \to \mathbb{Z}$ maps $x$ to the nearest integer. We quantize the weights using absmean quantization [24]. The scaling factor $\alpha_{\mathrm{W}}$ is defined as follows for this approach:

$$\alpha_{\mathrm{W}} = \frac{\mu}{\max(\mathcal{S})}, \quad \mu = \frac{1}{nm}\sum_i \sum_j |(\mathbf{W}_{\text{lin}})_{ij}|. \tag{4}$$

We also quantize the activations to $b$-bits by mapping them into the set $\mathcal{S}_{\mathrm{X}} = \{x \in \mathbb{Z} \mid -2^{b-1} \le x < 2^{b-1}\}$.:

$$\widetilde{\mathbf{X}}_{\text{lin}} = \mathrm{Clamp}\left(\mathrm{round}\left(\frac{\mathbf{X}_{\text{lin}}}{\alpha_{\mathrm{X}}}\right), \min(\mathcal{S}_{\mathrm{X}}), \max(\mathcal{S}_{\mathrm{X}})\right). \tag{5}$$

Different from weight quantization, we use absmax quantization [93] over the activations, and the scaling factor $\alpha_{\mathrm{X}}$ is defined as follows for this strategy:

$$\alpha_{\mathrm{X}} = \frac{||\mathbf{X}_{\text{lin}}||_\infty}{\max(\mathcal{S}_{\mathrm{X}})}. \tag{6}$$

$||\mathbf{X}_{\text{lin}}||_\infty$ refers to the $l_\infty$ norm of $\mathbf{X}_{\text{lin}}$. We use $b = 8$-bit and $b = 1$-bit representations to quantize the activations.

Through quantization, the linear layer in (1) is converted to the following:

$$\widetilde{\mathbf{Y}}_{\text{lin}} = \mathtt{Quant}(\mathbf{Y}_{\text{lin}}) = \widetilde{\mathbf{X}}_{\text{lin}}\widetilde{\mathbf{W}}_{\text{lin}}. \tag{7}$$

As a final adjustment, the output of the quantized linear layer is dequantized back to its original precision:

$$\hat{\mathbf{Y}}_{\text{lin}} = \mathtt{Dequant}(\widetilde{\mathbf{Y}}_{\text{lin}}) = \alpha_{\mathrm{W}}\alpha_{\mathrm{X}} \times \widetilde{\mathbf{Y}}_{\text{lin}}. \tag{8}$$

$\mathtt{Quant}(.)$ and $\mathtt{Dequant}(.)$ will be used to refer to the quantization and dequantization functions in (7) and (8), respectively. We use channel-wise scaling in both weight and activation quantization. We use the straight-through estimator (STE) [94] to approximate the gradient of quantized parameters, as the quantization function in (3) is non-differentiable and incompatible with backpropagation. We quantize all linear transformations within transformer layers, including the feed-forward, attention, and linear classifier layers, as well as the embedding parameters to low-bit precision.

*2) Quantized Transformer Blocks:* The quantized transformer receives $B$ raw sequences. The model tokenizes each sequence by mapping each element to tokens. Besides mapping the data entities, tokenization also adds special tokens to the sequence. The [CLS] token, placed at the start of the sequence, is used to aggregate information from sequence elements. The [SEP] token marks 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 $\mathbf{X} \in \mathbb{R}^{B\times M\times d}$ through its embedding layer, where $d$ is the embedding dimension. The embedding matrix is supplied to the quantized transformer model as input:

$$\mathbf{H}^{(0)} = \mathbf{X}. \tag{9}$$

The quantized transformer contains $L$ stacked blocks comprised of 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:

$$\mathbf{H}^{(l)} = \mathtt{QTB}\left(\mathbf{H}^{(l-1)}\right) \quad \forall l \in \{1, 2, \ldots, L\}. \tag{10}$$

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

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

$$\widetilde{\mathbf{Q}}_i^{(l)} = \widetilde{\mathbf{H}}_i^{(l-1)}\widetilde{\mathbf{W}}_{\mathbf{Q}_i}^{(l)}, \quad \widetilde{\mathbf{K}}_i^{(l)} = \widetilde{\mathbf{H}}_i^{(l-1)}\widetilde{\mathbf{W}}_{\mathbf{K}_i}^{(l)},$$
$$\widetilde{\mathbf{V}}_i^{(l)} = \widetilde{\mathbf{H}}_i^{(l-1)}\widetilde{\mathbf{W}}_{\mathbf{V}_i^{(l)}}^{(l)}, \tag{11}$$

where $\widetilde{\mathbf{W}}_{\mathbf{Q}_i^{(l)}}^{(l)}, \widetilde{\mathbf{W}}_{\mathbf{K}_i^{(l)}}^{(l)}, \widetilde{\mathbf{W}}_{\mathbf{V}_i^{(l)}}^{(l)} \in \mathbb{R}^{d\times\frac{d}{h}}$ are learnable quantized weights. MHSA computes attention scores at each head for all tokens within the sequence using the three matrices:

$$\mathtt{Att}_i\left(\widetilde{\mathbf{Q}}_i^{(l)}, \widetilde{\mathbf{K}}_i^{(l)}, \widetilde{\mathbf{V}}_i^{(l)}\right) = \mathtt{Softmax}\left(\frac{\widetilde{\mathbf{Q}}_i^{(l)}\widetilde{\mathbf{K}}_i^{(l)T}}{\sqrt{d/h}}\right)\widetilde{\mathbf{V}}_i^{(l)}, \tag{12}$$

The attention scores calculated at each head are then concatenated and passed through a quantized linear transformation:

$$\widetilde{\texttt{MHA}}\left(\widetilde{\mathbf{Q}}^{(l)}, \widetilde{\mathbf{K}}^{(l)}, \widetilde{\mathbf{V}}^{(l)}\right) = \texttt{Quant}([\text{head}_1^{(l)} \ldots \text{head}_h^{(l)}])\widetilde{\mathbf{W}}_o^{(l)}, \tag{13}$$

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

$$\mathbf{H}_{\text{att}}^{(l)} = \texttt{LN}\left(\mathbf{H}^{(l-1)} + \texttt{Dequant}\left(\widetilde{\texttt{MHA}}\left(\widetilde{\mathbf{Q}}^{(l)}, \widetilde{\mathbf{K}}^{(l)}, \widetilde{\mathbf{V}}^{(l)}\right)\right)\right). \tag{14}$$

Since they have negligible costs, layer normalizations and residual connections are maintained in full precision. After layer normalization, the signal is supplied to a quantized FFN:

$$\widetilde{\mathbf{H}}_{\text{FNN}}^{(l)} = \phi_{\text{T}}\left(\texttt{Quant}\left(\mathbf{H}_{\text{att}}^{(l)}\right)\widetilde{\mathbf{W}}_{\text{FFN}}^{(l)}\right), \tag{15}$$

where $\phi_{\text{T}}(.) = \text{GELU}(.)$ 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 a residual connection to obtain the $l$th QTB output:

$$\mathbf{H}^{(l)} = \texttt{LN}\left(\mathbf{H}_{\text{att}}^{(l)} + \texttt{Dequant}\left(\widetilde{\mathbf{H}}_{\text{FFN}}^{(l)}\right)\right). \tag{16}$$

Following (10), the output of the $L$th QTB $\mathbf{H}^{(L)} \in \mathbb{R}^{B\times M\times d}$ is obtained. After tokenization, the first entry of each sequence held the [CLS] token, which served to aggregate information from sequence elements. The QTBs produce an embedding for the [CLS] token, which we extract from $\mathbf{H}^{(L)}$ to construct sequence embeddings $\mathbf{U}_{\text{seq}} \in \mathbb{R}^{B\times d}$. $\mathbf{U}_{\text{seq}}$ is passed through a quantized linear classifier to obtain the final quantized transformer output:

$$\hat{\mathbf{Z}}_{\text{BitTrans}} = \texttt{Dequant}\left(\widetilde{\mathbf{U}}_{\text{seq}}\widetilde{\mathbf{W}}_{\text{cls}}\right), \tag{17}$$

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

$$\mathbf{P}_{\text{BitTrans}} = \texttt{Softmax}\left(\hat{\mathbf{Z}}_{\text{BitTrans}}\right). \tag{18}$$

*3) Integrating Quantization and Fine-Tuning:* We investigate the integration of quantization to transformer fine-tuning in two different phases.

1) *Quantization-Aware Training (QAT):* first quantizes pretrained model weights and activations, then fine-tunes the transformer after quantization [20], [22], [30].
2) *Post-Training Quantization (PTQ):* conducts fine-tuning using full-precision weights and quantizes transformer parameters and activations after the fine-tuning is completed [75], [95], [96].

PTQ is more practical to use, since it requires no retraining. However, because it does not optimize the model with quantized weights, low-bit PTQ often results in greater performance degradation compared to QAT.

## B. GNN Module

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

*1) Graph Generation:* To propagate information across text elements in language tasks, we construct a text graph from each given corpus. We place each word and document in the corpus on the nodes of a heterogeneous text graph. We construct the edges between different nodes through corpus statistics. We can represent the adjacency matrix $\mathbf{A} \in \mathbb{R}^{(N+V)\times(N+V)}$ of our graph structure over $N$ documents and $V$ unique words contained in the corpus through the following mathematical formulation:

$$\mathbf{A} = \begin{bmatrix} \mathbf{G}_{\text{w}} & \mathbf{D}^\top \\ \mathbf{D} & \mathbf{G}_{\text{s}} \end{bmatrix}. \tag{19}$$

$\mathbf{G}_{\text{w}} \in \mathbb{R}^{V\times V}$ defines the adjacency between word-word node pairs constructed using the PPMI metric, which captures corpus-level co-occurrence statistics among words. $\mathbf{D} \in \mathbb{R}^{N\times V}$ defines the connections between word and document nodes by computing the TF-IDF scores, which reflect the importance of words within individual documents. To construct the document-document similarity graph $\mathbf{G}_{\text{s}} \in \mathbb{R}^{N\times N}$, we compute TF-IDF representations $t_d \in \mathbb{R}^V$ for each document $d$. Then, we identify the $k = 25$ nearest neighbors $\mathcal{N}_k(d)$ of every document based on cosine similarity. Consequently, the adjacency among document-document node pairs is defined:

$$\mathbf{G}_{\text{s}}(d, d') = \begin{cases} \cos(t_d, t_{d'}) & \text{if } d' \in \mathcal{N}_k(d) \text{ or } d \in \mathcal{N}_k(d'), \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

*2) GNN Blocks:* The GNN module is constructed by stacking multiple message-passing layers that operate on graph-structured data. The feature update of a generic message-passing layer can be defined as the following:

$$\mathbf{F}^{(l)} = \phi_{\text{G}}\left(\mathbf{S}^{(l)}\mathbf{F}^{(l-1)}\mathbf{W}^{(l)}\right), \tag{21}$$

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 $\phi_{\text{G}}(.)$ is the nonlinear activation function used within GNN layers. We use graph convolutional layers as our message-passing components. For a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ 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. A graph convolutional layer formulates the graph structure encoding 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:

$$\mathbf{F}^{(l)} = \phi_{\text{G}}\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{F}^{(l-1)}\mathbf{W}^{(l)}\right). \tag{22}$$

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

We supply the sequence embeddings $\mathbf{U}_{\text{seq}}$ outputted by the quantized transformer as inputs to the graph module. The embeddings of sequence nodes, or the document node embeddings in the NLP context, are initialized using $\mathbf{U}_{\text{seq}} \in \mathbb{R}^{N\times d}$, and the

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

node embeddings of the $V$ unique tokens within the corpus vocabulary are initialized as $\mathbf{0} \in \mathbb{R}^{V \times d}$:

$$\mathbf{F}^{(0)} = \begin{pmatrix} \mathbf{U}_{\text{seq}} \\ \mathbf{0} \end{pmatrix} \in \mathbb{R}^{(N+V)\times d}. \tag{23}$$

These embeddings are processed by the GNN module that contains $K$ stacked graph convolutional layers defined in (22) as follows:

$$\mathbf{F}^{(K)} = \mathtt{GNN}(\mathbf{U}_{\text{seq}}). \tag{24}$$

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

$$\mathbf{P}_{\text{GNN}} = \mathtt{Softmax}(\mathbf{Z}_{\text{GNN}}). \tag{25}$$

### C. The Integration Module

Quantization holds a significant potential to reduce the memory and energy costs of transformers; however, it degrades the representational capabilities of transformer parameters. In addition, despite successfully representing the local context, transformers are poor at capturing global relationships across data entities extending beyond their context windows. To reduce the shortcomings of quantized transformers, BitTransGNN employs a GNN module that contains multiple message passing layers in conjunction with the BitTransformer during training, as can be seen visually in Fig. 1. The GNN operates on sequence embeddings produced by the BitTransformer to enrich the BitTransformer's output representations. The integration module synthesizes the outputs of the BitTransformer and the GNN to improve BitTransformer training.

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

$$\mathbf{P}_{\text{BitTransGNN}} = \lambda \mathbf{P}_{\text{GNN}} + (1-\lambda)\mathbf{P}_{\text{BitTrans}}, \tag{26}$$

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

$$\mathcal{L}_{\text{ce}} = -\frac{1}{|\mathcal{Y}_{\text{train}}|} \sum_{d\in\mathcal{Y}_{\text{train}}} \sum_{c=1}^{C} \mathbf{Y}_{dc} \ln(\mathbf{P}_{\text{BitTransGNN}})_{dc}, \tag{27}$$

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 $\mathcal{Y}_{\text{train}}$ 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 output:

$$\mathbf{Z}_{\text{BitTransGNN}} = \lambda \mathbf{Z}_{\text{GNN}} + (1-\lambda)\hat{\mathbf{Z}}_{\text{BitTrans}}, \tag{28}$$

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

$$\mathcal{L}_{\text{mse}} = \frac{1}{2|\mathcal{Y}_{\text{train}}|} \sum_{d\in\mathcal{Y}_{\text{train}}} (\mathbf{Y}_d - \mathbf{Z}_d)^2. \tag{29}$$

### D. Training Schemes

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

*1) 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 operates on these node embeddings. GNN only serves to improve the final outputs without an influence over the quantized transformer parameters.

*2) BitTransGNN$_{\text{Dynamic}}$:* The dynamic training scheme jointly trains the quantized transformer model and the GNN. While GCN is trained with full-batch gradient descent, transformer models cannot be trained in that manner due to memory constraints. Due to this distinction, we modified the training procedure to train the two models jointly. At each iteration, a mini-batch of document nodes is sampled from the text graph. Embeddings of these nodes are updated using the document embeddings $\mathbf{U}_{\text{seq}}$ outputted by the quantized transformer at that instance. GNN operates over the updated node features, and the quantized transformer produces outputs only for those document instances. Dynamic training also allows us to investigate whether the quantized model can find better parameters when trained jointly with a GNN.

### E. Inductive BitTransGNN Variants

BitTransGNN is based on an inherently transductive framework. This characteristic arises from the use of a GNN block with graph convolutional layers operating alongside a quantized transformer. While this property presents unique advantages in modeling global relationships, it also imposes a key limitation. Graph convolutional models learn transformations on node features conditioned on a fixed graph structure and cannot generate embeddings for unseen nodes without retraining. Consequently, since the adjacency matrix in (19) is defined over the entire corpus, the GNN block cannot infer on new data instances and must be retrained on an updated graph containing the additional word and document nodes. To extend our framework to inductive problem settings, we propose two methods that encapsulate the knowledge learned by the joint model into a quantized transformer model that can then operate independently in inductive scenarios, without incurring additional inference cost.

*1) Direct Separation (DS):* DS detaches the quantized transformer and its classifier from BitTransGNN and uses them directly for inference without any additional training. After DS, BitTransGNN output takes the following form:

$$\hat{\mathbf{Z}}_{\text{BitTrans}} = \mathtt{Dequant}\left(\widetilde{\mathbf{U}}_{\text{seq}}\widetilde{\mathbf{W}}_{\text{cls}}\right). \tag{30}$$

*2) Induction Through Knowledge Distillation (KD):* While DS allows inferring 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. We aim to enhance the performance of the standalone quantized transformer through knowledge distillation, a technique in which a smaller student model learns to replicate the output behavior of a larger, more expressive teacher model. To this end, we employ a teacher–student framework where a trained BitTransGNN variant serves as the teacher and an independent BitTransformer

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

as the student model. We apply offline distillation [59], [97] and only train the student quantized transformer. The intermediate layers of GNN and transformer are vastly different. Hence, we only establish correspondence between model output distributions and use a response-based distillation scheme [59], [98], [99], [100].

The student model is trained on a combination of distillation loss and ground truth loss. We compute the distillation loss $\mathcal{L}_{\mathrm{d}}$ over the logits of the student and the teacher models, and the ground-truth loss $\mathcal{L}_{\mathtt{truth}}$ over the student model outputs and the ground-truth labels. The combined loss function $\mathcal{L}$ is formulated as following:

$$\mathcal{L} = \alpha_{\mathrm{d}}\mathcal{L}_{\mathrm{d}} + (1-\alpha_{\mathrm{d}})\mathcal{L}_{\mathtt{truth}}, \tag{31}$$

where $\alpha_{\mathrm{d}} \in [0,1]$ is a hyperparameter that determines the influence of the distillation loss over the joint loss function.

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. Since we want the student model to mimic the teacher, we want to minimize the KL divergence between the output distributions of the two models. For the output probability distributions of the student model $\mathbf{P}_{\mathrm{S}}$ and the teacher model $\mathbf{P}_{\mathrm{T}}$, the KL divergence is computed over $C$ different classes as follows:

$$\mathtt{KL}(\mathbf{P}_{\mathrm{T}}, \mathbf{P}_{\mathrm{S}}) = \sum_{b=1}^{B}\sum_{c=1}^{C}(\mathbf{P}_{\mathrm{T}})_{bc}\log\frac{(\mathbf{P}_{\mathrm{S}})_{bc}}{(\mathbf{P}_{\mathrm{T}})_{bc}}. \tag{32}$$

In KD, we use a modified softmax function with an additional temperature hyperparameter $\tau$ to convert model outputs into probabilities. $\tau \geq 1$ softens the output distributions to provide more information during training. Softmax with temperature is defined as follows for some matrix $\mathbf{Z}$:

$$\mathtt{Softmax}_{\tau}(\mathbf{Z}) = \mathtt{Softmax}\left(\frac{\mathbf{Z}}{\tau}\right). \tag{33}$$

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

$$\mathbf{P}_{bc} = \mathtt{Softmax}_{\tau}(\mathbf{Z}_{bc}) = \sum_{b=1}^{B}\sum_{c=1}^{C}\frac{\exp((\mathbf{Z})_{bc}/\tau)}{\sum_{k=1}^{C}\exp((\mathbf{Z})_{bk}/\tau)}. \tag{34}$$

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

$$\mathcal{L}_{\mathrm{d}} = \frac{\tau^2}{B}\mathtt{KL}(\mathbf{P}_{\mathrm{T}}, \mathbf{P}_{\mathrm{S}}), \tag{35}$$

where $\mathbf{P}_{\mathrm{T}} = \mathtt{Softmax}_{\tau}(\mathbf{Z}_{\mathrm{T}})$ and $\mathbf{P}_{\mathrm{S}} = \mathtt{Softmax}_{\tau}(\mathbf{Z}_{\mathrm{S}})$ for some temperature $\tau$. The distillation loss is scaled by the factor $\tau^2$ to compensate for the scaling effect of the temperature on the gradients during backpropagation. In regression tasks, the distillation loss is defined by the MSE loss between the teacher and the student model outputs:

$$\mathcal{L}_{\mathrm{d}} = \frac{1}{2B}\sum_{b=1}^{B}((\mathbf{Z}_{\mathrm{T}})_b - (\mathbf{Z}_{\mathrm{S}})_b)^2. \tag{36}$$

TABLE II
NUMBER OF BITS REQUIRED TO STORE BITTRANSFORMER AND BITTRANSGNN PARAMETERS IN DIFFERENT QUANTIZATION SCENARIOS

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

## F. *Efficiency Analysis*

We conduct a theoretical analysis to display the potential efficiency advantages of our architecture in terms of memory overhead and energy consumption. Below, we summarize the analysis, and further details on calculations are presented in the Supplementary Material.

*1) Memory Efficiency:* The memory required to store the parameters of BERT-based baseline models and BitTransGNN variants is listed in Table II 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. The fully quantized configuration $(b_l, b_e) = (1,1)$ requires 32 times less parameter memory compared to its full-precision counterpart. Despite the additional GNN parameters, the joint BitTransGNN models remain efficient, as their overhead is negligible relative to a full-precision transformer. As BitTransGNN variants have comparable sizes to the BitTransformer, the proposed joint framework and the distillation procedure in Section III-E2 can be trained in a memory-efficient manner.

*2) Energy Efficiency:* Table III summarizes the theoretical energy costs of baseline methods and BERT-based BitTransGNN variants across multiple bit-precisions and data types. The reported values estimate inference energy for a full dataset pass, accounting for the arithmetic operations in each forward pass. Although the results focus on inference, we also distinguish between BitTransGNN$_{\mathrm{Static}}$ and BitTransGNN$_{\mathrm{Dynamic}}$ by their training costs: the static variant employs the GNN once per epoch, whereas the dynamic variant uses it at every batch. During inference, however, both can operate at the same cost as BitTransGNN$_{\mathrm{Static}}$ since the quantized transformer remains unchanged. Energy costs are dataset-dependent; to outline the worst case, our analysis uses the 20NG dataset, which is the most computationally demanding. 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 yields about a 1.4$\times$ reduction, as these representations allow for replacing costly multiplication operations with additions and subtractions. BitTransGNN$_{\mathrm{Static}}$ requires negligible additional energy cost relative to quantized transformer baselines, while BitTransGNN$_{\mathrm{Dynamic}}$ incurs about 14% more energy expense, since the GCN has to operate over the full-graph structure at every batch. Despite this additional

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

TABLE III
ENERGY CONSUMPTION BY ARITHMETIC OPERATIONS PER EPOCH FOR BITTRANSGNN VARIANTS AND BASELINE METHODS OVER 7 NM PROCESSORS

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

overhead, all BitTransGNN$_{\text{Dynamic}}$ variants remain more efficient than full-precision baselines. Processing data with integer-based representations can further amplify these savings, as the arithmetic operations of quantized transformers consume over 16$\times$ less energy with 32-bit integers and over 100$\times$ less with 8-bit integers. These efficiency gains motivate our adoption of integer-type quantization in BitTransGNN variants.

## IV. EXPERIMENTAL RESULTS[1]

We describe the datasets in Section IV-A and outline the experimental setup in Section IV-B. Section IV-C reports model performances against baselines, highlights gains from integrating PTQ methods into BitTransGNN, and supports the theoretical efficiency analyses with hardware measurements. Section IV-D presents ablation and inference time analyses.

### A. Datasets

We assessed our method on various language tasks, including sentence topic classification, sentiment analysis, textual entailment, and semantic similarity. We conducted experiments over nine benchmark datasets. We used *20 Newsgroups (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 [101].

### B. Experimental Setup

We integrate base-sized pre-trained BERT and RoBERTa models available in Hugging Face Transformers library [102] into the BitTransGNN architecture. We conduct experiments with transformers quantized to 1-bit, 1.58-bit, and 2.32-bit parameter precision using the corresponding quantization sets listed in Table I. We quantize and fine-tune the pre-trained transformer models using the strategies presented in Section III-A3 before integrating them into BitTransGNN. We ran each experiment with five different seeds and reported the mean and standard deviation of the results.

We compare our methods against full-precision and quantized transformer models. We use full-precision BERT [2] and RoBERTa [3] transformers, as well as their quantized variants that are fine-tuned independently (without GNN integration), as our baseline models. In our tables, we label the 32-bit full-precision transformer baselines as BERT (FP32) and RoBERTa (FP32), and we denote quantized transformer models fine-tuned without GNN integration as BitTransformer (QAT). We also report the performance of Outlier Suppression (OS) [71] and Outlier Suppression+ (OS+) [72], applied to the BERT backbone, as advanced post-training quantization baselines to assess how joint transformer–GNN training compares with more sophisticated PTQ techniques. Further details on the experimental setup, dataset statistics, baseline models, and the ranges of hyperparameters that we searched through can be found in the Supplementary Material.

### C. Results and Evaluations

*1) Main Results:* We present the results of transductive and inductive BitTransGNN methods trained on the listed language tasks and compare their performances against the baseline models in Table IV. In the table, the transformer and BitTransGNN model instances are grouped according to their backbone and parameter bit-precision. For post-training quantization (PTQ) baselines (OS and OS+), the same bit-precision is applied to both weights and activations, and this bit-precision is shown alongside their labels. For each instance, we report the performance of the joint model variants BitTransGNN$_{\text{Static}}$ and BitTransGNN$_{\text{Dynamic}}$, as well as the inductive variants DS and KD, which we refer to as BitTransGNN-DS and BitTransGNN-KD, respectively. DS is applied only to BitTransGNN$_{\text{Dynamic}}$, as BitTransGNN$_{\text{Static}}$ does not train the quantized transformer. During KD, we select the best-performing instance from BitTransGNN$_{\text{Static}}$ and BitTransGNN$_{\text{Dynamic}}$ on the validation set as the teacher, and the student BitTransformer is quantized to the same bit-precision. To assess model performances, we report Matthew's Correlation for the CoLA dataset, the F1 score for the MRPC dataset, and the Pearson Correlation for the STS-B dataset. For the rest of the tasks, we report the accuracy of each method.

The joint BitTransGNN methods significantly outperform quantized transformer baselines that are trained without a GNN's collaboration, while requiring only a minor amount of additional overhead. BERT-based 1-bit BitTransGNN$_{\text{Dynamic}}$ reports 88.48% mean accuracy on the 20NG dataset with $\lambda = 0.55$, 6.40% higher than its 1-bit BitTransformer counterpart, and outperforming the full-precision transformer baseline. Quantization causes severe degradation on some of the more challenging GLUE tasks, such as CoLA. The 2.32-bit RoBERTa-based BitTransformer model reports a 25.45%

[1]Our codes are available at https://github.com/koc-lab/BitTransGNN

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
COMPARISON OF BITTRANSGNN METHODS WITH FULL-PRECISION AND QUANTIZED TRANSFORMER BASELINES. THE BEST RESULTS ARE BOLDFACED, THE SECOND-BEST ARE UNDERLINED, AND THE THIRD-BEST ARE ITALICIZED

| | **Size** | **Energy** | **20NG** | **MR** | **R8** | **R52** | **Ohsumed** | **CoLA** | **MRPC** | **RTE** | **STS-B** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | MB | J | Accuracy | Accuracy | Accuracy | Accuracy | Accuracy | Matthew's | F1 | Accuracy | Pearson |
| **PTQ Baselines** | | | | | | | | | | | |
| OS (8-bit) | 109.6 | 356.0 | 84.56 | 85.34 | 97.49 | 96.22 | 71.56 | $\underline{58.48}$ | 72.61 | 57.40 | 86.33 |
| OS (6-bit) | 82.2 | 356.0 | 82.58 | 83.34 | 97.40 | 93.57 | 69.85 | 56.01 | 72.18 | 63.18 | 87.30 |
| OS (4-bit) | 54.9 | 356.0 | 77.78 | 68.60 | 96.16 | 90.50 | 62.63 | 16.78 | 53.74 | 49.10 | 62.44 |
| OS+ (8-bit) | 109.6 | 356.0 | 84.48 | 85.40 | 97.72 | 96.18 | 71.36 | 56.01 | 72.80 | 55.60 | 86.30 |
| OS+ (6-bit) | 82.2 | 356.0 | 76.14 | 76.73 | 95.89 | 87.89 | 65.45 | 41.52 | 66.49 | 50.54 | 85.11 |
| OS+ (4-bit) | 54.9 | 356.0 | 55.39 | 57.63 | 93.19 | 79.36 | 52.26 | 08.07 | 51.59 | 47.29 | 40.02 |
| **BitTransGNN** | | | | | | | | | | | |
| **BERT** | | | | | | | | | | | |
| **1-bit** | | | | | | | | | | | |
| BERT (FP32) | 438.0 | 356.0 | 85.18 ± 0.41 | $\underline{85.88 \pm 0.54}$ | 97.84 ± 0.49 | 96.14 ± 0.25 | 70.88 ± 0.80 | *58.08 ± 2.33* | *72.89 ± 1.34* | *65.99 ± 2.69* | $\underline{89.55 \pm 0.12}$ |
| BitTransformer (QAT) | 13.8 | 165.7 | 82.08 ± 0.45 | $\underline{76.49 \pm 0.56}$ | 97.36 ± 0.52 | 94.89 ± 0.38 | 64.18 ± 0.62 | 16.26 ± 2.34 | 51.57 ± 1.83 | 47.15 ± 0.41 | $\underline{72.88 \pm 1.52}$ |
| BitTransGNN$_{\text{Static}}$ | 20.3 | 165.8 | 85.97 ± 0.38 | 77.96 ± 0.16 | 97.89 ± 0.10 | 95.77 ± 0.11 | 66.98 ± 0.31 | 15.33 ± 0.62 | 47.75 ± 0.73 | 53.29 ± 1.57 | 72.13 ± 0.02 |
| BitTransGNN$_{\text{Dynamic}}$ | 20.3 | 215.7 | $\underline{88.48 \pm 0.05}$ | 78.20 ± 0.72 | 97.73 ± 0.03 | 95.90 ± 0.28 | 69.86 ± 0.23 | 18.34 ± 2.09 | 47.90 ± 0.97 | 49.03 ± 1.64 | 76.51 ± 0.26 |
| BitTransGNN − DS | 13.8 | 165.7 | 82.75 | 76.59 | 97.35 | 95.64 | 64.75 | 17.45 | 40.73 | 54.15 | 75.35 |
| BitTransGNN − KD | 13.8 | 165.7 | 83.86 ± 0.28 | 78.14 ± 0.85 | 97.42 ± 0.32 | 95.71 ± 0.35 | 66.43 ± 1.99 | 15.49 ± 1.31 | 50.94 ± 1.11 | 51.62 ± 1.47 | 77.94 ± 0.75 |
| **1.58-bit** | | | | | | | | | | | |
| BERT (FP32) | 438.0 | 356.0 | 85.18 ± 0.41 | $\underline{85.88 \pm 0.54}$ | 97.84 ± 0.49 | 96.14 ± 0.25 | 70.88 ± 0.80 | *58.08 ± 2.33* | *72.89 ± 1.34* | *65.99 ± 2.69* | $\underline{89.55 \pm 0.12}$ |
| BitTransformer (QAT) | 21.8 | 243.5 | 82.66 ± 0.39 | $\underline{78.55 \pm 0.58}$ | 97.57 ± 0.33 | 95.16 ± 0.36 | 66.30 ± 0.93 | 21.92 ± 3.96 | 49.30 ± 0.82 | 51.62 ± 2.94 | $\underline{77.81 \pm 0.86}$ |
| BitTransGNN$_{\text{Static}}$ | 28.2 | 165.8 | 87.89 ± 0.17 | 79.05 ± 0.07 | 97.87 ± 0.04 | 94.92 ± 0.06 | 70.17 ± 1.00 | 22.44 ± 0.01 | 50.11 ± 0.56 | 55.60 ± 1.55 | 79.17 ± 0.02 |
| BitTransGNN$_{\text{Dynamic}}$ | 28.2 | 215.7 | 88.33 ± 0.18 | 81.16 ± 0.43 | 97.85 ± 0.11 | 96.44 ± 0.16 | 68.41 ± 0.71 | 31.73 ± 2.47 | 48.57 ± 2.74 | 52.27 ± 3.56 | 78.44 ± 0.50 |
| BitTransGNN − DS | 21.8 | 165.7 | 82.58 | 79.29 | 97.99 | 96.30 | 68.32 | 36.03 | 45.42 | 54.15 | 78.41 |
| BitTransGNN − KD | 21.8 | 165.7 | 84.89 ± 0.33 | 80.87 ± 0.35 | 97.56 ± 0.26 | 96.11 ± 0.16 | 68.76 ± 0.49 | 32.42 ± 1.90 | 49.88 ± 2.57 | 52.78 ± 1.23 | 83.43 ± 0.34 |
| **2.32-bit** | | | | | | | | | | | |
| BERT (FP32) | 438.0 | 356.0 | 85.18 ± 0.41 | $\underline{85.88 \pm 0.54}$ | 97.84 ± 0.49 | 96.14 ± 0.25 | 70.88 ± 0.80 | *58.08 ± 2.33* | *72.89 ± 1.34* | *65.99 ± 2.69* | $\underline{89.55 \pm 0.12}$ |
| BitTransformer (QAT) | 31.9 | 165.7 | 84.31 ± 0.25 | $\underline{83.83 \pm 0.18}$ | 97.59 ± 0.32 | 95.74 ± 0.55 | 68.56 ± 1.12 | 50.19 ± 2.21 | 68.08 ± 2.74 | 60.00 ± 1.98 | $\underline{88.00 \pm 0.23}$ |
| BitTransGNN$_{\text{Static}}$ | 38.4 | 243.6 | 88.23 ± 0.12 | 84.79 ± 0.12 | 97.60 ± 0.05 | $\underline{96.61 \pm 0.03}$ | 70.37 ± 0.43 | 46.03 ± 0.35 | $\underline{73.27 \pm 0.37}$ | 61.16 ± 0.79 | 88.58 ± 0.01 |
| BitTransGNN$_{\text{Dynamic}}$ | 38.4 | 293.6 | **88.78 ± 0.11** | 84.66 ± 0.51 | $\underline{98.18 \pm 0.29}$ | **96.73 ± 0.23** | *71.78 ± 0.45* | 54.00 ± 0.68 | 72.69 ± 1.28 | 65.85 ± 1.32 | *89.14 ± 0.15* |
| BitTransGNN − DS | 31.9 | 243.5 | 85.42 | 84.55 | 97.53 | *96.46* | 58.50 | 55.79 | 65.71 | $\underline{67.15}$ | 88.96 |
| BitTransGNN − KD | 31.9 | 243.5 | 85.42 ± 0.52 | 84.64 ± 0.44 | 97.73 ± 0.07 | 96.24 ± 0.15 | $\underline{71.91 \pm 0.22}$ | 53.80 ± 1.60 | 71.78 ± 1.70 | 65.34 ± 1.51 | 89.04 ± 0.07 |
| **RoBERTa** | | | | | | | | | | | |
| **1-bit** | | | | | | | | | | | |
| RoBERTa (FP32) | 498.6 | 356.0 | 83.43 ± 0.62 | **88.89 ± 0.39** | 97.67 ± 0.55 | 96.00 ± 0.41 | 71.67 ± 0.90 | **59.87 ± 1.65** | **80.28 ± 0.59** | **76.97 ± 2.74** | **90.48 ± 0.35** |
| BitTransformer (QAT) | 15.7 | 165.7 | 78.35 ± 0.36 | 76.12 ± 0.42 | 97.42 ± 0.52 | 94.35 ± 0.76 | 59.38 ± 1.51 | 08.56 ± 1.37 | 49.21 ± 1.68 | 48.52 ± 1.46 | 34.67 ± 0.70 |
| BitTransGNN$_{\text{Static}}$ | 22.2 | 165.8 | 87.03 ± 0.22 | 78.90 ± 0.14 | 97.75 ± 0.04 | 95.04 ± 0.02 | 67.24 ± 0.33 | 10.56 ± 0.53 | 50.33 ± 0.01 | 48.66 ± 0.16 | 33.70 ± 0.06 |
| BitTransGNN$_{\text{Dynamic}}$ | 22.2 | 215.7 | 85.99 ± 0.64 | 77.97 ± 0.79 | 96.37 ± 0.72 | 93.37 ± 0.37 | 66.77 ± 0.74 | 9.78 ± 0.85 | 50.29 ± 2.29 | 48.52 ± 2.02 | 37.26 ± 2.12 |
| BitTransGNN − DS | 15.7 | 165.7 | 75.45 | 75.49 | 96.53 | 92.64 | 49.79 | 14.31 | 50.00 | 55.60 | 29.58 |
| BitTransGNN − KD | 15.7 | 165.7 | 78.75 ± 0.83 | 76.79 ± 0.40 | 96.55 ± 0.36 | 94.27 ± 0.24 | 61.85 ± 0.58 | 11.30 ± 1.37 | 50.05 ± 1.84 | 47.87 ± 0.98 | 31.52 ± 1.12 |
| **1.58-bit** | | | | | | | | | | | |
| RoBERTa (FP32) | 498.6 | 356.0 | 83.43 ± 0.62 | **88.89 ± 0.39** | 97.67 ± 0.55 | 96.00 ± 0.41 | 71.67 ± 0.90 | **59.87 ± 1.65** | **80.28 ± 0.59** | **76.97 ± 2.74** | **90.48 ± 0.35** |
| BitTransformer (QAT) | 24.8 | 165.7 | 80.11 ± 1.00 | 79.63 ± 0.27 | 97.55 ± 0.34 | 95.30 ± 0.40 | 63.78 ± 0.93 | 18.06 ± 2.63 | 50.79 ± 1.88 | 51.19 ± 2.52 | 55.64 ± 2.23 |
| BitTransGNN$_{\text{Static}}$ | 31.2 | 165.8 | 86.56 ± 0.20 | 80.35 ± 0.06 | *98.12 ± 0.11* | 95.52 ± 0.07 | 67.68 ± 0.17 | 17.71 ± 0.01 | 48.83 ± 1.32 | 53.36 ± 0.30 | 57.84 ± 0.03 |
| BitTransGNN$_{\text{Dynamic}}$ | 31.2 | 215.7 | 87.44 ± 0.26 | 80.69 ± 0.38 | 97.75 ± 0.11 | 95.60 ± 0.28 | 68.79 ± 2.34 | 18.00 ± 0.84 | 48.14 ± 1.28 | 51.84 ± 2.77 | 57.72 ± 0.92 |
| BitTransGNN − DS | 24.8 | 165.7 | 78.96 | 78.59 | 97.58 | 95.68 | 65.22 | 16.20 | 47.66 | 53.07 | 56.44 |
| BitTransGNN − KD | 24.8 | 165.7 | 81.36 ± 0.34 | 80.15 ± 0.73 | 97.91 ± 0.27 | 95.77 ± 0.19 | 67.86 ± 0.19 | 23.02 ± 1.82 | 50.63 ± 1.09 | 49.10 ± 1.84 | 59.41 ± 1.58 |
| **2.32-bit** | | | | | | | | | | | |
| RoBERTa (FP32) | 498.6 | 356.0 | 83.43 ± 0.62 | **88.89 ± 0.39** | 97.67 ± 0.55 | 96.00 ± 0.41 | 71.67 ± 0.90 | **59.87 ± 1.65** | **80.28 ± 0.59** | **76.97 ± 2.74** | **90.48 ± 0.35** |
| BitTransformer (QAT) | 36.3 | 243.5 | 81.97 ± 1.13 | 83.58 ± 0.75 | 97.68 ± 0.27 | 95.11 ± 0.47 | 65.66 ± 1.24 | 35.42 ± 3.61 | 52.65 ± 0.93 | 50.54 ± 1.75 | 77.54 ± 1.83 |
| BitTransGNN$_{\text{Static}}$ | 42.8 | 243.6 | 87.81 ± 0.25 | 84.61 ± 0.02 | 97.57 ± 0.01 | 95.50 ± 0.03 | 68.47 ± 1.22 | 41.64 ± 0.01 | 51.86 ± 1.14 | 51.84 ± 0.20 | 79.42 ± 0.02 |
| BitTransGNN$_{\text{Dynamic}}$ | 42.8 | 293.6 | *88.37 ± 0.28* | *85.71 ± 0.33* | **98.19 ± 0.11** | 95.98 ± 0.28 | **72.14 ± 0.68** | 53.66 ± 1.83 | 51.93 ± 2.79 | 51.91 ± 1.23 | 82.76 ± 0.57 |
| BitTransGNN − DS | 36.3 | 243.5 | 81.73 | 84.89 | 97.76 | 96.30 | 67.15 | 48.58 | 48.83 | 47.65 | 83.35 |
| BitTransGNN − KD | 36.3 | 243.5 | 82.80 ± 0.19 | 85.22 ± 0.36 | 97.91 ± 0.17 | 96.22 ± 0.13 | 70.68 ± 0.41 | 53.22 ± 1.29 | 50.89 ± 3.17 | 51.12 ± 2.38 | 87.04 ± 0.57 |

decrease in Matthew's correlation metric. The corresponding BitTransGNN$_{\text{Dynamic}}$ model significantly restores the loss in performance, outperforming the BitTransformer by 18.24%. BitTransGNN$_{\text{Static}}$ proves almost as equally effective as its dynamic equivalent despite its much lower energy cost. As an example, the BERT-based 2.32-bit BitTransGNN$_{\text{Static}}$ surpasses the BitTransformer performance by 5.19% on the MRPC task, matching full-precision model accuracy.

Inductive BitTransGNN variants also show significant improvements over quantized transformer baselines, despite having identical inference costs. DS improves quantized transformer performance across nearly all tasks. The 2.32-bit BERT-based BitTransGNN-DS outperforms a BitTransformer model by 7.15% on the RTE dataset. Benefiting from additional supervision, KD excels even beyond DS in most tasks and consistently enhances quantized transformer training. The 2.32-bit RoBERTa-based BitTransGNN-KD surpasses the corresponding BitTransformer by 9.50% on the STS-B task. These results demonstrate that the proposed methods effectively transfer global information into quantized transformers, enhancing their representational capabilities. Moreover, both inductive variants encapsulate a notable amount of the joint method's knowledge representation within a standalone quantized transformer, achieving competitive performance with full-precision models despite the representational limitations imposed by quantization. We performed a paired one-sided t-test for statistical significance. Most of the test results are statistically significant with $p$ values of $p < 0.005$ and even $p < 0.001$ for some cases, with only a few marginal or non-significant cases. Further details of statistical tests are presented in the Supplementary Material.

*2) Cost-Performance Tradeoff:* To evaluate the cost-performance tradeoff, we provide a visual comparison of BitTransGNN variants and baseline models on the 20NG dataset in terms of accuracy and theoretical memory and energy costs in Fig. 2. We also include the BERTGCN model from [54] in these plots as a full-precision reference for BitTransGNN$_{\text{Dynamic}}$. The joint BitTransGNN models generally occupy the regions of the plots that correspond to lower memory and higher performance, as they outperform BitTransformers and require substantially less memory than full-precision baselines.

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

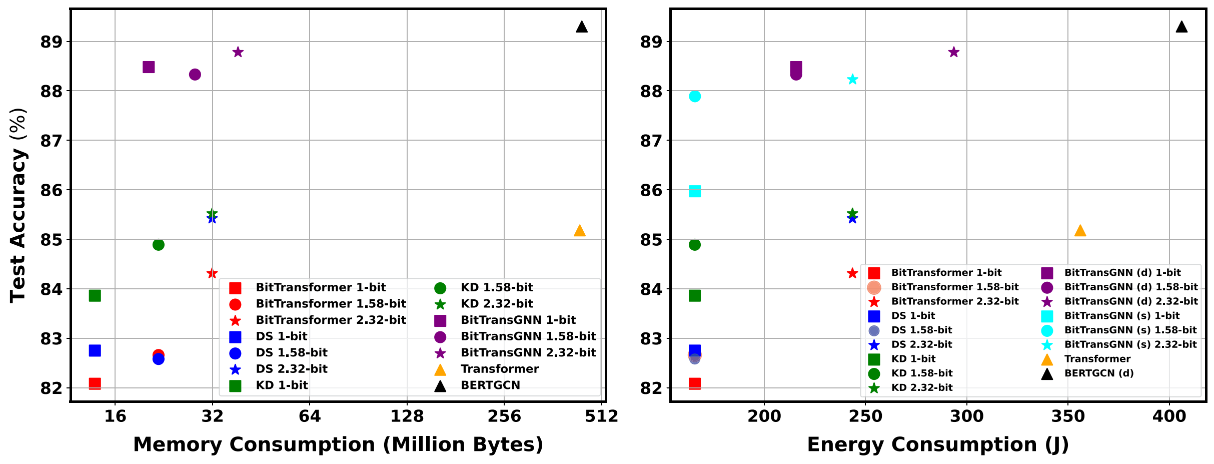

Fig. 2. Performance of BitTransGNN variants and baseline methods on the 20NG dataset, showing test accuracy versus memory consumption (left) and energy consumption (right). The x-axis in the left plot is shown on a $\log_2$ scale to better distinguish between quantized and full-precision memory consumption.

Inductive variants KD and DS methods are placed slightly lower, as they lose some performance while trying to confine the knowledge of BitTransGNN within a standalone quantized transformer. Despite such a drop, they still perform close to the full-precision transformer with much smaller memory costs. Examining the model performances against energy consumptions per epoch, it can be observed that the transformer models quantized to low-bit precision consume significantly less energy during inference. DS and KD methods benefit from this efficiency while maintaining competitive performance against full-precision transformers. BitTransGNN$_{\text{Static}}$ outperforms BitTransformers with negligible additional energy costs, whereas BitTransGNN$_{Dynamic}$ achieves higher accuracy, although at a higher energy cost due to the GNN's use in every batch. Despite this drawback, all BitTransGNN$_{Dynamic}$ models still consume less energy than full-precision transformers across all datasets. Further comparisons of energy costs using int32 and int8 data types, as well as visualizations of BitTransGNN's output representations, are provided in the Supplementary Material.

*3) Integrating PTQ Methods Into BitTransGNN:* BitTransGNN performs comparably to full-precision transformers and PTQ baselines quantized to higher bit-widths. However, one might question whether the gains attained from the GNN integration could be achieved simply through more sophisticated quantization techniques. To examine this, we incorporated the quantized transformers produced by OS and OS+ into the BitTransGNN framework and evaluated whether our proposed method could further improve their performance. The results, summarized in Table V, show that both transductive and inductive BitTransGNN variants consistently enhance the accuracy of OS and OS+ baselines across all quantization settings. The 8-bit variants even match or exceed the full-precision transformer performance. The 6-bit PTQ baselines suffer a more notable decrease in model performance due to lower bit-width, and the BitTransGNN framework effectively compensates for the degradations stemming from quantization. These findings demonstrate that the benefits of BitTransGNN extend beyond improvements in quantization methodology, highlighting its

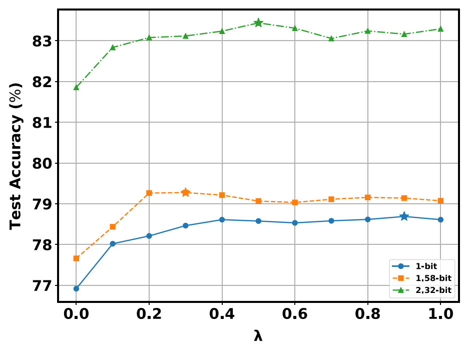

Fig. 3. Accuracy of BERT-based BitTransGNN$_{\text{Static}}$ on the MR dataset against varying values of $\lambda$ listed for different quantization rates.

flexibility and compatibility with existing quantization techniques for achieving superior overall performance.

*4) Hyperparameter Analysis:* To evaluate the impact of the integration module on BitTransGNN performance, we analyze the effect of the hyperparameter $\lambda$, which controls the balance between BitTransformer and GNN outputs. Fig. 3 presents BitTransGNN$_{\text{Static}}$ performance on the MR dataset across different quantization rates and $\lambda$ values. The 1-bit BitTransGNN relies most heavily on the GNN, with about 85% of its predictions derived from it, whereas higher-bit models draw only 30–50% of their outputs from the GNN. Although not a strict rule, a clear trend emerges: lower-precision transformers benefit from larger $\lambda$, while models with higher bit-widths depend less on the GNN. This trend suggests that BitTransGNN's dependence on the GNN varies with the representational capacity of the quantized transformer. The sensitivity to $\lambda$ is further reflected in the DS method. While it generally improves BitTransformer performance, it performs worse on the Ohsumed task for 1-bit RoBERTa- and 2.32-bit BERT-based BitTransGNN. Both operate with $\lambda = 1.0$, where the model relies solely on the GNN for predictions. This example highlights the importance of maintaining transformer participation at the prediction stage if the GNN is omitted after training. It should be noted that dataset

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

TABLE V
PERFORMANCE OF BITTRANSGNN METHODS APPLIED TO BERT-BASED PTQ BASELINES. THE BEST RESULTS ARE BOLDFACED

| | **Size** | **Energy** | **20NG** | **MR** | **R8** | **R52** | **Ohsumed** | **CoLA** | **MRPC** | **RTE** | **STS-B** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | MB | J | Accuracy | Accuracy | Accuracy | Accuracy | Accuracy | Matthew’s | F1 | Accuracy | Pearson |
| **Full-Precision** | | | | | | | | | | | |
| BERT (FP32) | 438.0 | 356.0 | 85.18 ± 0.41 | 85.88 ± 0.54 | 97.84 ± 0.49 | 96.14 ± 0.25 | 70.88 ± 0.80 | 58.08 ± 2.33 | **72.89 ± 1.34** | 65.99 ± 2.69 | 89.55 ± 0.12 |
| **BitTransGNN** | | | | | | | | | | | |
| **OS** | | | | | | | | | | | |
| **8-bit** | | | | | | | | | | | |
| PTQ Baseline | 109.6 | 356.0 | 84.56 | 85.34 | 97.49 | 96.22 | 71.56 | **58.48** | 72.61 | 57.40 | 86.33 |
| $\text{BitTransGNN}_{\text{Static}}$ | 116.1 | 356.1 | 88.69 ± 0.13 | **86.33 ± 0.11** | 98.01 ± 0.05 | 96.18 ± 0.17 | 71.95 ± 0.12 | 56.86 ± 0.16 | 72.39 ± 0.46 | 63.47 ± 0.47 | 87.41 ± 0.06 |
| $\text{BitTransGNN}_{\text{Dynamic}}$ | 116.1 | 406.1 | 88.51 ± 0.20 | 85.98 ± 0.28 | 98.02 ± 0.15 | **96.97 ± 0.16** | 71.82 ± 0.26 | 57.56 ± 2.44 | 72.62 ± 2.15 | 64.33 ± 4.38 | 89.72 ± 0.12 |
| BitTransGNN − DS | 109.6 | 356.0 | 85.33 | 85.34 | 97.81 | 96.69 | 70.24 | 57.02 | 68.54 | 67.87 | **89.85** |
| BitTransGNN − KD | 109.6 | 356.0 | 85.80 ± 0.17 | 85.86 ± 0.21 | 97.72 ± 0.16 | 96.81 ± 0.06 | 71.92 ± 0.36 | 56.79 ± 1.11 | 71.19 ± 1.57 | 67.29 ± 0.94 | 89.67 ± 0.14 |
| **6-bit** | | | | | | | | | | | |
| PTQ Baseline | 109.6 | 356.0 | 82.58 | 83.34 | 97.40 | 93.57 | 69.85 | 56.01 | 72.18 | 63.18 | 87.30 |
| $\text{BitTransGNN}_{\text{Static}}$ | 116.1 | 356.1 | 87.65 ± 0.18 | 84.15 ± 0.09 | 97.73 ± 0.08 | 93.42 ± 0.24 | 71.98 ± 0.76 | 55.04 ± 0.14 | 72.51 ± 0.14 | 61.16 ± 1.50 | 88.97 ± 0.01 |
| $\text{BitTransGNN}_{\text{Dynamic}}$ | 116.1 | 406.1 | 88.59 ± 0.08 | 85.85 ± 0.37 | 97.97 ± 0.32 | 96.70 ± 0.10 | 71.75 ± 0.67 | 56.46 ± 2.24 | 72.58 ± 1.56 | 65.27 ± 2.17 | 89.62 ± 0.09 |
| BitTransGNN − DS | 109.6 | 356.0 | 85.37 | 85.09 | 97.62 | 96.50 | 70.57 | 55.54 | 71.71 | 65.71 | 89.54 |
| BitTransGNN − KD | 109.6 | 356.0 | 85.40 ± 0.30 | 85.41 ± 0.47 | 97.76 ± 0.11 | 96.85 ± 0.16 | 71.70 ± 0.55 | 56.46 ± 1.74 | 72.71 ± 0.99 | 66.35 ± 1.90 | 89.51 ± 0.07 |
| **OS+** | | | | | | | | | | | |
| **8-bit** | | | | | | | | | | | |
| PTQ Baseline | 109.6 | 356.0 | 84.48 | 85.40 | 97.72 | 96.18 | 71.36 | 56.01 | 72.80 | 55.60 | 86.30 |
| $\text{BitTransGNN}_{\text{Static}}$ | 116.1 | 356.1 | **88.77 ± 0.19** | 86.27 ± 0.21 | 97.95 ± 0.02 | 96.10 ± 0.08 | **72.00 ± 0.36** | 55.07 ± 0.31 | 71.86 ± 0.31 | 60.29 ± 0.72 | 86.81 ± 0.07 |
| $\text{BitTransGNN}_{\text{Dynamic}}$ | 116.1 | 406.1 | 88.67 ± 0.15 | 85.93 ± 0.35 | 98.07 ± 0.23 | 96.93 ± 0.16 | 71.74 ± 0.49 | 57.55 ± 1.13 | 72.35 ± 1.84 | 67.44 ± 1.48 | 89.58 ± 0.12 |
| BitTransGNN − DS | 109.6 | 356.0 | 85.28 | 84.64 | 97.90 | 96.85 | 69.35 | 58.67 | 72.63 | **69.68** | 89.67 |
| BitTransGNN − KD | 109.6 | 356.0 | 85.79 ± 0.11 | 85.70 ± 0.16 | 97.50 ± 0.17 | 96.74 ± 0.18 | 71.86 ± 0.37 | 57.82 ± 1.39 | 72.86 ± 1.25 | 66.43 ± 1.42 | 89.56 ± 0.09 |
| **6-bit** | | | | | | | | | | | |
| PTQ Baseline | 82.2 | 356.0 | 76.14 | 76.73 | 95.89 | 87.89 | 65.45 | 41.52 | 66.49 | 50.54 | 85.11 |
| $\text{BitTransGNN}_{\text{Static}}$ | 88.7 | 356.1 | 86.98 ± 0.31 | 79.38 ± 0.11 | 97.21 ± 0.09 | 90.03 ± 0.38 | 70.28 ± 0.73 | 41.53 ± 0.06 | 67.46 ± 0.20 | 60.36 ± 1.00 | 86.34 ± 0.03 |
| $\text{BitTransGNN}_{\text{Dynamic}}$ | 88.7 | 406.1 | 88.38 ± 0.24 | 84.93 ± 0.35 | **98.11 ± 0.42** | 96.69 ± 0.13 | 71.77 ± 0.24 | 54.01 ± 1.95 | 72.77 ± 1.19 | 67.08 ± 1.48 | 89.03 ± 0.27 |
| BitTransGNN − DS | 82.2 | 356.0 | 85.16 | 85.23 | 97.72 | 96.57 | 69.75 | 52.49 | 73.33 | 64.98 | 89.09 |
| BitTransGNN − KD | 82.2 | 356.0 | 85.36 ± 0.14 | 85.30 ± 0.38 | 97.72 ± 0.08 | 96.65 ± 0.26 | 71.32 ± 0.66 | 53.35 ± 1.17 | 70.52 ± 2.99 | 65.85 ± 1.78 | 89.01 ± 0.13 |

TABLE VI
INFERENCE TIME, POWER, AND ENERGY CONSUMPTION OF FULL-PRECISION AND QUANTIZED (INT8) TRANSFORMER VARIANTS, WITH AND WITHOUT GNN INTEGRATION, MEASURED ON CPU AND GPU. QUANTIZED MODELS EMPLOY SPECIALIZED MATRIX-MULTIPLICATION KERNELS

| | **MRPC** | | | **Ohsumed** | | | **CoLA** | | | **20NG** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | CPU | GPU | | CPU | GPU | | CPU | GPU | | CPU | GPU | |
| **Model** | Time (s) ↓ | Power (W) ↓ | Energy (J) ↓ | Time (s) ↓ | Power (W) ↓ | Energy (J) ↓ | Time (s) ↓ | Power (W) ↓ | Energy (J) ↓ | Time (s) ↓ | Power (W) ↓ | Energy (J) ↓ |
| **Full-Precision** | | | | | | | | | | | | |
| Transformer | 26.04 | 293.83 | 548.60 | 21.35 | 242.21 | 403.16 | 51.80 | 231.98 | 959.43 | 71.09 | 251.69 | 1357.16 |
| Transformer + GNN | 26.24 | 292.89 | 553.98 | 22.01 | 240.16 | 423.64 | 51.87 | 231.91 | 961.56 | 74.36 | 248.20 | 1445.66 |
| **Quantized (INT8)** | | | | | | | | | | | | |
| Transformer | 19.85 | 219.37 | 452.86 | 16.15 | 197.85 | 351.79 | 41.19 | 197.73 | 864.30 | 55.50 | 203.65 | 1162.61 |
| Transformer + GNN | 20.03 | 219.37 | 458.10 | 16.81 | 198.12 | 372.02 | 41.26 | 197.74 | 866.45 | 58.82 | 203.55 | 1250.94 |

characteristics also affect optimal $\lambda$, as no single value yields the best results across all tasks.

*5) Hardware Measurements:* To complement the theoretical analysis in Section III-F assessing the computational efficiency of BitTransGNN, we conducted hardware measurements. We employed PyTorch’s dynamic quantization backend [103] to evaluate inference time on CPU and *bitsandbytes* [68], [69] to measure energy consumption and average power usage on GPU. Results for full-precision and quantized Transformer variants, both standalone and integrated with a GNN, are reported in Table VI. Using BERT as the transformer backbone, we measured each metric over a full forward pass on the test set. Quantization reduced CPU inference time by up to 23% on the MRPC dataset. It has lowered average power usage on the GPU by more than 20% in all tasks. These results validate our theoretical estimates, showing that quantization effectively offsets the additional computations introduced by the GNN. While highlighting the potential of BitTransGNN, current toolkits remain limited to 8-bit parameter representations. Future hardware developments will be key to realizing the benefits of lower-bit precisions and further exploiting the efficiency advantages of BitTransGNN [68], [69], [70].

## *D. Ablation Studies*

We conducted a set of ablation experiments to examine the significance of different design choices on the effectiveness of our proposed models.

*1) Graph Construction:* To assess the contribution of different sub-components within the proposed graph structure of (19), we conduct an ablation study comparing BitTransGNN variants trained with different adjacency configurations. In each setting, specific edge types in the adjacency matrix are masked. We evaluate four configurations: the full adjacency ($\mathbf{A} = \mathbf{A}^{(\text{full})}$); the graph without document–document edges ($\mathbf{A} = \mathbf{A}^{(-\mathbf{G}_\text{S})}$); the graph without word–word edges ($\mathbf{A} = \mathbf{A}^{(-\mathbf{G}_\text{W})}$); and the graph containing only document nodes ($\mathbf{A} = \mathbf{G}_\text{S}$). Table VII reports both model performance and total inference duration over the entire dataset for each configuration. For text classification tasks with large vocabularies, such as 20NG and Ohsumed, omitting word nodes leads to significantly faster inference. However, models trained with graphs only containing document nodes exhibit the lowest performance in these tasks, as the absence of word–word and word–document relations restricts information flow and creates a bottleneck at document nodes. This highlights a trade-off between efficiency and representational richness

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
PERFORMANCE AND INFERENCE DURATION OF BERT-BASED BITTRANSGNN VARIANTS TRAINED WITH DIFFERENT ADJACENCY MATRICES. FOR BITTRANSGNN VARIANTS, THE INFERENCE TIME OF THE TRANSFORMER'S EMBEDDING EXTRACTION PHASE AND THE PER-EPOCH COMPUTATIONS OF THE JOINT MODEL ARE REPORTED AS SEPARATED BY "+". "**" DENOTES OUR MAIN CONFIGURATION. THE BEST RESULTS ARE HIGHLIGHTED IN BOLD

| | 20NG | | MR | | Ohsumed | | STS-B | |
|---|---|---|---|---|---|---|---|---|
| **Model** | Duration (s) | Accuracy | Duration (s) | Accuracy | Duration (s) | Accuracy | Duration (s) | Pearson |
| $\mathbf{A} = \mathbf{0}$ (No GNN) | | | | | | | | |
| BitTransformer | 22.19 | 82.08 ± 0.45 | 12.56 | 76.49 ± 0.56 | 8.70 | 64.18 ± 0.62 | 8.56 | 72.88 ± 1.52 |
| $\mathbf{A} = \mathbf{A}^{(\text{full})}$** | | | | | | | | |
| BitTransGNN$_{\text{Static}}$ | 22.19 + 0.090 | 85.97 ± 0.38 | 12.56 + 0.0072 | 77.96 ± 0.16 | 8.70 + 0.026 | 66.98 ± 0.31 | 8.56 + 0.0049 | 72.13 ± 0.02 |
| BitTransGNN$_{\text{Dynamic}}$ | 22.19 + 74.22 | **88.48 ± 0.05** | 12.56 + 14.64 | 78.20 ± 0.72 | 8.70 + 14.50 | **69.86 ± 0.23** | 8.56 + 20.99 | 76.51 ± 0.26 |
| $\mathbf{A} = \mathbf{A}^{(-\mathbf{G}_{\text{s}})}$ | | | | | | | | |
| BitTransGNN$_{\text{Static}}$ | 22.19 + 0.088 | 86.16 ± 0.37 | 12.56 + 0.0063 | 78.03 ± 0.19 | 8.70 + 0.025 | 66.85 ± 0.75 | 8.56 + 0.0044 | 72.16 ± 0.02 |
| BitTransGNN$_{\text{Dynamic}}$ | 22.19 + 73.01 | 88.34 ± 0.29 | 12.56 + 14.34 | **78.39 ± 0.31** | 8.70 + 14.30 | 69.76 ± 0.50 | 8.56 + 20.94 | 76.83 ± 0.70 |
| $\mathbf{A} = \mathbf{A}^{(-\mathbf{G}_{\text{w}})}$ | | | | | | | | |
| BitTransGNN$_{\text{Static}}$ | 22.19 + 0.020 | 86.41 ± 0.29 | 12.56 + 0.0045 | 77.82 ± 0.29 | 8.70 + 0.0049 | 67.75 ± 0.68 | 8.56 + 0.0033 | 72.32 ± 0.03 |
| BitTransGNN$_{\text{Dynamic}}$ | 22.19 + 33.14 | 88.61 ± 0.22 | 12.56 + 13.77 | 78.09 ± 0.36 | 8.70 + 9.62 | 69.48 ± 0.34 | 8.56 + 20.24 | 77.77 ± 0.73 |
| $\mathbf{A} = \mathbf{G}_{\text{s}}$ | | | | | | | | |
| BitTransGNN$_{\text{Static}}$ | 22.19 + 0.0041 | 85.58 ± 0.12 | 12.56 + 0.0024 | 77.47 ± 0.08 | 8.70 + 0.0021 | 65.13 ± 0.65 | 8.56 + 0.0022 | 72.08 ± 0.01 |
| BitTransGNN$_{\text{Dynamic}}$ | 22.19 + 23.53 | 84.45 ± 0.44 | 12.56 + 12.96 | 77.48 ± 0.90 | 8.70 + 8.93 | 66.65 ± 0.54 | 8.56 + 20.09 | **78.11 ± 0.40** |

TABLE VIII
PERFORMANCE OF BERT-BASED TRANSFORMERS AND BITTRANSGNN VARIANTS UNDER DIFFERENT QUANTIZATION CONFIGURATIONS. W$\{b_l\}$E$\{b_e\}$A$\{b_a\}$ INDICATES THE BIT-PRECISION OF LINEAR TRANSFORMATION ($b_l$), EMBEDDING PARAMETERS ($b_e$) AND ACTIVATIONS ($b_a$). "**" DENOTES OUR MAIN CONFIGURATION

| | Size | Energy | 20NG | MR | Ohsumed | RTE |
|---|---|---|---|---|---|---|
| **Model** | MB | J | Accuracy | Accuracy | Accuracy | Accuracy |
| **W32E32A32** | | | | | | |
| Transformer | 438.0 | 356.0 | 85.18 ± 0.41 | 85.88 ± 0.54 | 70.88 ± 0.80 | 65.99 ± 2.69 |
| **W1E32A8** | | | | | | |
| BitTransformer | 106.2 | 165.7 | 81.49 ± 0.46 | 76.92 ± 0.56 | 63.34 ± 1.00 | 50.47 ± 3.08 |
| BitTransGNN$_{\text{Static}}$ | 112.7 | 165.8 | 86.26 ± 0.27 | 78.64 ± 0.10 | 69.36 ± 0.50 | 54.01 ± 0.41 |
| BitTransGNN$_{\text{Dynamic}}$ | 112.7 | 215.7 | 88.65 ± 0.08 | 79.27 ± 0.22 | 68.90 ± 1.20 | 53.43 ± 1.28 |
| BitTransGNN − DS | 106.2 | 165.7 | 82.97 | 77.12 | 66.41 | 49.10 |
| BitTransGNN − KD | 106.2 | 165.7 | 84.77 ± 0.23 | 78.47 ± 0.45 | 69.46 ± 0.30 | 51.77 ± 1.82 |
| **W1E1A8** ** | | | | | | |
| BitTransformer | 13.8 | 165.7 | 82.08 ± 0.45 | 76.49 ± 0.56 | 64.18 ± 0.62 | 47.15 ± 0.41 |
| BitTransGNN$_{\text{Static}}$ | 20.3 | 165.8 | 85.97 ± 0.38 | 77.96 ± 0.16 | 66.98 ± 0.31 | 53.29 ± 1.57 |
| BitTransGNN$_{\text{Dynamic}}$ | 20.3 | 215.7 | 88.48 ± 0.05 | 78.20 ± 0.72 | 69.86 ± 0.23 | 49.03 ± 1.64 |
| BitTransGNN − DS | 13.8 | 165.7 | 82.75 | 76.59 | 64.75 | 54.15 |
| BitTransGNN − KD | 13.8 | 165.7 | 83.86 ± 0.28 | 78.14 ± 0.85 | 66.43 ± 1.99 | 51.62 ± 1.47 |
| **W1E1A1** | | | | | | |
| BitTransformer | 13.8 | 165.7 | 72.13 ± 1.74 | 76.03 ± 0.41 | 45.19 ± 2.15 | 49.82 ± 2.28 |
| BitTransGNN$_{\text{Static}}$ | 20.3 | 165.8 | 81.87 ± 0.77 | 78.09 ± 0.09 | 53.46 ± 1.60 | 54.37 ± 1.04 |
| BitTransGNN$_{\text{Dynamic}}$ | 20.3 | 215.7 | 81.51 ± 0.92 | 76.83 ± 0.30 | 53.22 ± 0.68 | 51.26 ± 1.61 |
| BitTransGNN − DS | 13.8 | 165.7 | 72.54 | 74.51 | 41.41 | 50.18 |
| BitTransGNN − KD | 13.8 | 165.7 | 74.57 ± 0.36 | 76.50 ± 0.46 | 49.09 ± 0.44 | 51.34 ± 1.15 |

in graph construction. As a contrasting case, on the STS-B task, BitTransGNN models achieve higher performance when the word nodes are eliminated from the text-graph; however, the corresponding reduction in inference duration is much less pronounced than in other tasks.

*2) Quantization Configuration:* To evaluate the sensitivity of different BitTransGNN components to quantization, we conduct an ablation study across multiple bit-width configurations. We denote each configuration as W$\{b_l\}$E$\{b_e\}$A$\{b_a\}$, where $b_l$, $b_e$, and $b_a$ are the bit precision of linear transformation weights, embedding parameters, and activation signals, respectively. The results are presented in Table VIII. Embedding quantization causes only minor performance loss in moderately sized transformers such as BERT. Consequently, the main configuration in our study (W1E1A8) performs comparably to the mixed-precision setup (W1E32A8) while requiring substantially less memory. Conversely, the fully binarized configuration (W1E1A1) minimizes arithmetic costs but suffers a sharp degradation in accuracy. Although BitTransGNN variants still outperform their corresponding transformer baselines under extreme quantization, the performance gap to full-precision models becomes more pronounced. Overall, these observations support the choice of the W1E1A8 configuration as a balanced trade-off between model performance and computational efficiency.

## V. DISCUSSION

The core inspiration of BitTransGNN stemmed from the conjecture that transformer models and GNNs capture different characteristics of data: transformers model relatively local contextual semantics, while GNNs are successful in representing global relationships that can be extracted from the entirety of data. 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) [104] on the final logits of the BitTransformer, $\hat{\mathbf{Z}}_{\text{BitTrans}}$, and the GNN module, $\mathbf{Z}_{\text{GNN}}$ . To conduct RSA, we create a similarity matrix among the means of each class representation by computing pair-wise distances within the two logits. We stacked the similarity matrices of the two methods to construct similarity profiles and implemented Principal Component Analysis (PCA) on these profiles to compare the representational focus of the BitTransformer and the GNN. We display the first two principal components (PCs) in Fig. 4. The PCs of the BitTransformer and GNN output representations are clearly segregated into two clusters, which are easily separable, as they are situated far apart along the axis corresponding to the first PC. This shows that the BitTransformer and the GNN modules focus on highly distinctive features while constructing their outputs, providing insight into the success of the BitTransGNN and how the GNN module compensates for the shortcomings of transformer models to improve quantized transformer training.

We employed graph-convolutional layers in our GNN blocks due to their effectiveness in modeling relationships between corpus elements and their suitability for the datasets considered in this work. A potential drawback of this design is that GCN layers can become inefficient when applied to very large or dense graphs. In our experiments, the 20NG dataset represented the largest graph structure, where full-graph training remained tractable. However, scaling to graphs with millions of nodes would pose challenges since standard GCNs operate over the

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

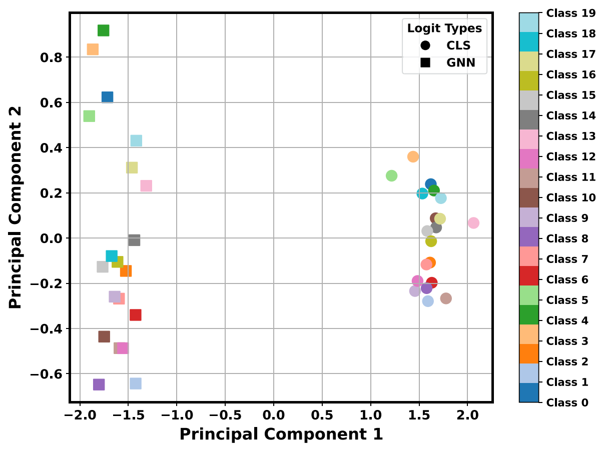

Fig. 4. PCA analysis on the similarity profiles of the logits of BitTransformer and GNN modules within a BERT-based 1-bit BitTransGNN$_{\text{Dynamic}}$ model trained on the 20NG task. The importance profiles are computed using RSA.

entire adjacency matrix and do not support mini-batch training. In such settings, established strategies such as neighborhood sampling [38], subgraph sampling [105], or cluster-based sampling [106] can be used to enable efficient training through partitioned batches, thereby avoiding full-graph propagation at each iteration. Enabling mini-batch training can also mitigate the computational overhead of BitTransGNN$_{\text{Dynamic}}$, which in its current form depends on a full-batch GCN and therefore reprocesses the entire graph at each training iteration.

The use of graph-convolutional layers imposes a transductive characteristic on BitTransGNN, since the GCN block operates over a fixed graph structure. In small-scale settings where all nodes are known during training, the GCN can exploit full graph connectivity, including unlabeled nodes, to learn stronger embeddings and often outperform sampling-based methods. Alternatives such as GraphSAGE [38] provide inductive capability and scale well to very large graphs, but their reliance on local neighborhood sampling can reduce performance on smaller graphs where full-batch training is feasible. For these reasons, we used the GCN model instead of an inductive GNN alternative. To address the limitations of transductive inference, we proposed methods in Section III-E that adapt BitTransGNN to inductive scenarios. Where the joint setting is strictly required to be inductive, the GCN can be replaced by alternative GNN modules that support inductive inference, without altering the overall design.

While our experiments focus on sequence-level classification and regression, BitTransGNN is not inherently restricted to these, and can be adapted to generative tasks such as next-token prediction. To align the GNN model with the causal nature of autoregressive prediction, graph construction should enforce directed edges that connect each token only to its preceding context. In this way, the quantized transformer block can continue to capture local dependencies within its self-attention window, while the GNN block can provide complementary information beyond the self-attention window, thereby enriching long-range modeling for generation. Building on BitTransGNN's demonstrated compatibility with advanced PTQ methods, future extensions can integrate recent quantization techniques designed for generative tasks, including SpinQuant [73] and LeanQuant [74]. However, very large-scale corpora would introduce additional efficiency challenges, which can be mitigated by the sampling-based techniques discussed above. We leave exploration of generative applications to future work, but emphasize that the collaborative framework is applicable to both discriminative and generative tasks.

## VI. Conclusion

We introduced BitTransGNN to improve quantized transformers by integrating a GNN during training. BitTransGNN compensates for the performance loss arising from quantization by supplying the transformer with global context, which is modeled by a GNN module, that attention has difficulty capturing. To extend BitTransGNN beyond its transductive nature, we also proposed two inductive variants that encapsulate the joint model's knowledge into standalone quantized transformers, enabling inference with zero additional cost. Its inductive variants further encapsulate the knowledge learned by the joint BitTransGNN into standalone quantized transformers, achieving notable gains over quantized baselines with zero additional inference cost.

Through theoretical analysis, we demonstrated the strong potential of low-bit quantization to reduce transformer memory and energy costs, and showed that BitTransGNN can leverage these efficiency gains while outperforming quantized baselines. Hardware measurements supported these findings, even under current kernel constraints, and future advances in quantization toolkits and hardware-specific kernels are expected to further enhance the practical efficiency of our design.

Across multiple sequence-level language benchmarks, we experimentally demonstrated that BitTransGNN consistently outperforms quantized transformers trained without GNN assistance while introducing only minor additional overhead, and achieves performance comparable to full-precision baselines. Moreover, by incorporating PTQ baselines into our framework, we demonstrated that BitTransGNN yields complementary improvements beyond those achievable through state-of-the-art quantization techniques alone.

The synergy between transformers and GNNs in capturing different aspects of data underlies our approach to mitigating the cost-accuracy trade-off arising from quantization, which we experimentally validated using the RSA technique.

Our findings highlight a promising new direction, showing that modeling global relationships across data entities can enhance the expressivity of quantized transformers without compromising efficiency. Future work may extend this framework to larger-scale settings and generalize its application to generative tasks and to modalities beyond language.

## References

[1] A. Vaswani et al., “Attention is all you need,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30, pp. 5998–6008.
[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., Vol. 1 (Long Short Papers)*, 2019, pp. 4171–4186.
[3] Y. Liu et al., “RoBERTa: A Robustly optimized BERT pretraining approach,” 2019, *arXiv:1907.11692*.

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

[4] I. Beltagy, M. E. Peters, and A. Cohan, "LongFormer: The long-document transformer," 2020, *arXiv:2004.05150*.

[5] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Representations*, 2021.

[6] B. Jiang, K. Zhao, and J. Tang, "RGTransformer: Region-graph transformer for image representation and few-shot classification," *IEEE Signal Process. Lett.*, vol. 29, pp. 792–796, 2022.

[7] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "ViViT: A video vision transformer," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 6836–6846.

[8] B. De Weerdt, Y. C. Eldar, and N. Deligiannis, "Deep unfolding transformers for sparse recovery of video," *IEEE Trans. Signal Process.*, vol. 72, pp. 1782–1796, 2024.

[9] A. Akman, Q. Sun, and B. W. Schuller, "Improving audio explanations using audio language models," *IEEE Signal Process. Lett.*, vol. 32, pp. 741–745, 2025.

[10] N. Chen, S. Watanabe, J. Villalba, P. Żelasko, and N. Dehak, "Non-autoregressive transformer for speech recognition," *IEEE Signal Process. Lett.*, vol. 28, pp. 121–125, 2021.

[11] J. Kaplan et al., "Scaling laws for neural language models," 2020, *arXiv:2001.08361*.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI, blog*, 2019. [Online]. Available: https://openai.com/research/language-unsupervised

[13] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?," in *Proc. ACM Conf. Fairness, Accountability, Transparency*, 2021, pp. 610–623.

[14] M. C. Rillig, M. Ågerstrand, M. Bi, K. A. Gould, and U. Sauerland, "Risks and benefits of large language models for the environment," *Environ. Sci. Tech.*, vol. 57, no. 9, pp. 3464–3466, 2023.

[15] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 1877–1901.

[16] A. Faiz et al., "LLMCarbon: Modeling the end-to-end carbon footprint of large language models," in *Proc. Int. Conf. Comput. Representations*, 2024, pp. 1–15.

[17] E. Strubell, A. Ganesh, and A. Mccallum, "Energy and policy considerations for deep learning in NLP," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 3645–3650.

[18] H. Wang et al., "BitNet: Scaling 1-bit transformers for large language models," 2023, *arXiv:2310.11453*.

[19] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Proc. Low-Power Comput. Vis.*, 2022, pp. 291–326.

[20] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, vol. 28, pp. 3123–3131.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 187, pp. 1–30, 2018.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, vol. 29, pp. 4107–4115.

[23] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient AI applications," in *Proc. Int. Joint Conf. Neural Netw.*, 2017, pp. 2547–2554.

[24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[25] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, "DeepShift: Towards multiplication-less neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops*, 2021, pp. 2359–2368.

[26] J. Tian, C. Fang, H. Wang, and Z. Wang, "BEBERT: Efficient and robust binary ensemble BERT," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process..*, 2023, pp. 1–5.

[27] H. Qin et al., "BiBERT: Accurate fully binarized BERT," in *Proc. Int. Conf. Learn. Representations*, 2022.

[28] H. Bai et al., "BinaryBERT: Pushing the limit of BERT quantization," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. Natural Lang. Process. (Vol. 1, Long Papers)*, 2021, pp. 4334–4348.

[29] Y. He et al., "BiViT: Extremely compressed binary vision transformers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2023, pp. 5651–5663.

[30] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: Quantized 8Bit BERT," in *Proc. 5th Workshop Energy Efficient Mach. Learn. Cogn. Comput.*, 2019, pp. 36–39.

[31] S. Shen et al., "Q-BERT: Hessian based ultra low precision quantization of BERT," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, pp. 8815–8821.

[32] S. Ma et al., "The era of 1-bit LLMs: All large language models are in 1.58 bits," 2024, *arXiv:2402.17764*.

[33] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[34] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*.

[35] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proc. Int. Conf. Learn. Representations*, 2014.

[36] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. 34th Int. Conf. Mach. Learn.*, D. Precup and Y. W. Teh, Eds., Aug. 2017, vol. 70, pp. 1263–1272.

[37] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018.

[38] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30, pp. 1024–1034.

[39] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, Jun. 2019, pp. 6861–6871.

[40] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, Jan. 2019.

[41] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, vol. 70, pp. 933–941.

[42] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, 1982.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error-propagation," *Biometrika*, vol. 71, no. 599–607, 1986, Art. no. 6.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[45] K. Cho et al., "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Oct. 2014, pp. 1724–1734.

[46] A. Gu, K. Goel, and C. Re, "Efficiently modeling long sequences with structured state spaces," in *Proc. Int. Conf. Learn. Representations*, 2021.

[47] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," in *Proc. COLM,*, 2023.

[48] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," 2020, *arXiv:2006.04768*.

[49] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *Proc. Int. Conf. Learn. Representations*, 2020.

[50] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira, "Perceiver: General perception with iterative attention," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 4651–4664.

[51] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2019, vol. 33, pp. 7370–7377.

[52] R. Ragesh, S. Sellamanickam, A. Iyer, R. Bairi, and V. Lingam, "HeteGCN: Heterogeneous graph convolutional networks for text classification," in *Proc. ACM Int. Conf. Web Search Data Mining.*, 2021, pp. 860–868.

[53] H. Zhang and J. Zhang, "Text graph transformer for document classification," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2020, pp. 8322–8327.

[54] Y. Lin et al., "BertGCN: Transductive text classification by combining GNN and BERT," in *Proc. Findings Assoc. Comput. Linguistics*, 2021, pp. 1456–1462.

[55] A. C. Aras, T. Alikaşifoğlu, and A. Koç, "Graph receptive transformer encoder for text classification," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 10, pp. 347–359, 2024.

[56] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, vol. 29, pp. 3844–3852.

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.

[57] V. N. Vapnik, *Statistical Learning Theory*, vol. 2. Hoboken, NJ, USA: Wiley, 1998.

[58] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Proc. Adv. Neural Inf. Proc. Syst.*, 2003, vol. 16, pp. 321–328.

[59] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. Adv. Neural Inf. Process. Syst., Workshop Deep Learn.*, 2015.

[60] C. Bucilu, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2006, pp. 535–541.

[61] V. Sanh, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," in *Proc. Neural Inf. Proc. Syst.*, 2019.

[62] 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," in *Proc. Adv. Neural Inf. Proc. Syst.*, 2020, vol. 33, pp. 5776–5788.

[63] X. Jiao et al., "TinyBERT: Distilling BERT for natural language understanding," in *Proc. Findings Assoc. Comput. Linguistics*, T. Cohn, Y. He, and Y. Liu, Eds., Nov. 2020, pp. 4163–4174.

[64] Z. Liu et al., "BiT: Robustly binarized multi-distilled transformer," in *Proc. 36th Int. Conf. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 14303–14316.

[65] J. Kim, Y. Bhalgat, J. Lee, C. Patel, and N. Kwak, "QKD: Quantization-aware knowledge distillation," 2019, *arXiv:1911.12491*.

[66] P. Chen, B. Zhuang, and C. Shen, "FATNN: Fast and accurate ternary neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2021, pp. 5219–5228.

[67] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of BLOOM, a 176b parameter language model," *J. Mach. Learn. Res.*, vol. 24, no. 1, pp. 11990–12004, 2023.

[68] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," in *Proc. Int. Conf. Learn. Representations*, 2022.

[69] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-Bit matrix multiplication for transformers at scale," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 30318–30332, 2022.

[70] G. Park et al., "LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models," in *Proc. Int. Conf. Learn. Representations*, 2024.

[71] X. Wei et al., "Outlier suppression: Pushing the limit of low-bit transformer language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 17402–17414.

[72] X. Wei et al., "Outlier suppression: Accurate quantization of large language models by equivalent and effective shifting and scaling," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2023, pp. 1648–1665.

[73] Z. Liu et al., "Spinquant: LLM Quantization with learned rotations," in *Proc. Int. Conf. Learn. Representations*, 2025.

[74] T. Zhang and A. Shrivastava, "LeanQuant: Accurate and scalable large language model quantization with loss-error-aware grid," in *Proc. Int. Conf. Learn. Representations*, 2025.

[75] E. Frantar, S. Ashkboos, T. Hoefler, and D.-A. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," in *Proc. Int. Conf. Learn. Representations*, 2023.

[76] Z. Yuan, Y. Shang, and Z. Dong, "PB-LLM: Partially binarized large language models," in *Proc. Int. Conf. Learn. Representations*, 2024.

[77] W. Liu, Y. Xiao, D. Zeng, H. Zhao, W. Chen, and M. Zhang, "Mixed-precision graph neural quantization for low bit large language models," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2025, pp. 1–5.

[78] L. Li, H. Feng, and B. Hu, "Evaluation and prediction of mixed-precision quantization for edge model by graph transformer," in *Proc. 4th Asia Conf. Algorithms, Comput. Mach. Learn.*, 2025, pp. 1–8.

[79] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, vol. 55, no. 6, pp. 1–28, 2022.

[80] L. Wu et al., "Graph neural networks for natural language processing: A survey," *Foundations Trends Mach. Learn.*, vol. 16, no. 2, pp. 119–328, 2023.

[81] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, "Graph convolutional encoders for syntax-aware neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, M. Palmer, R. Hwa, and S. Riedel, Eds., Sep. 2017, pp. 1957–1967.

[82] D. Beck, G. Haffari, and T. Cohn, "Graph-to-sequence learning using gated graph neural networks," in *Proc. 56th Annu. Meeting Assoc. Comput. Linguistics*, L. Pap., I. Gurevych, and Y. Miyao, Eds., Jul. 2018, pp. 273–283.

[83] D. Wang, P. Liu, Y. Zheng, X. Qiu, and X. Huang, "Heterogeneous graph neural networks for extractive document summarization," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds., Jul. 2020, pp. 6209–6219.

[84] P. Fernandes, M. Allamanis, and M. Brockschmidt, "Structured neural summarization," in *Proc. Int. Conf. Learn. Representations*, 2019.

[85] A. C. Aras, T. Alikaşifoğlu, and A. Koç, "Text-RGNNs: Relational modeling for heterogeneous text graphs," *IEEE Signal Process. Lett.*, vol. 31, pp. 1955–1959, 2024.

[86] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 793–803.

[87] Z. Chen et al., "Exploring the potential of large language models (LLMs) in learning on graphs," *ACM SIGKDD Explorations Newslett.*, vol. 25, no. 2, pp. 42–61, 2024.

[88] C. Qian, H. Tang, Z. Yang, H. Liang, and Y. Liu, "Can Large language models empower molecular property prediction?," 2023, *arXiv:2307.07443*.

[89] J. Tang et al., "GraphGPT: Graph instruction tuning for large language models," in *Proc. 47th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2024, pp. 491–500.

[90] T. Alikaşifoğlu, A. Aras, and A. Koc, "VISPool: Enhancing transformer encoders with vector visibility graph neural networks," in *Proc. Findings Assoc. Comput. Linguistics*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand, Aug. 2024, pp. 2547–2556.

[91] E. Koç, A. C. Aras, T. Alikaşifoğlu, and A. Koç, "GraphTeacher: Transductive fine-tuning of encoders through graph neural networks," *IEEE Trans. Artif. Intell.*, early access, Oct. 31, 2025, doi: 10.1109/TAI.2025.3627514.

[92] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern. Recognit.*, 2018, pp. 2704–2713.

[93] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "GPT3.int8(): 8-bit matrix multiplication for transformers at scale," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 30318–30332.

[94] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.

[95] W. Huang et al., "BiLLM: Pushing the limit of post-training quantization for LLMs," in *Proc. 41st Int. Conf. Mach. Learn.*, 2024, pp. 20023–20042.

[96] H. Bai, L. Hou, L. Shang, X. Jiang, I. King, and M. R. Lyu, "Towards efficient post-training quantization of pre-trained language models," in *Proc. Adv. Neural Inf. Proc. Syst.*, 2022, vol. 35, pp. 1405–1418.

[97] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Proc. Adv. Neural Inf. Proc. Syst.*, 2017, vol. 30, pp. 742–751.

[98] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *Proc. Int. Conf. Learn. Representations*, 2015.

[99] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *ICLR*, 2016, *arXiv:1612.03928*.

[100] J. Kim, S. Park, and N. Kwak, "Paraphrasing complex network: Network compression via factor transfer," in *Proc. Adv. Neural Inf. Proc. Syst.*, 2018, vol. 31, pp. 2765–2774.

[101] 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 *Proc. EMNLP Workshop BlackboxNLP, Anal. Interpreting Neural Netw. NLP*, G. Linzen Chrupała and A. Alishahi, Eds., Nov. 2018, pp. 353–355.

[102] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proc. Conf. Empirical Methods Natural Lang. Process.: Syst. Demonstrations*, 2020, pp. 38–45.

[103] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Proc. Syst.*, 2019, vol. 32, pp. 8026–8037.

[104] N. Kriegeskorte, M. Mur, and P. A. Bandettini, "Representational similarity analysis-connecting the branches of systems neuroscience," *Front. Syst. Neurosci.*, vol. 2, 2008, Art. no. 249.

[105] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "GraphSAINT: Graph sampling based inductive learning method," in *Proc. Int. Conf. Learn. Representations*, 2020.

[106] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 257–266.

Authorized licensed use limited to: ULAKBIM UASL - Bilkent University. Downloaded on February 23,2026 at 17:16:30 UTC from IEEE Xplore. Restrictions apply.