EdgeMIN:A Two Stage Compression Pipeline for Efficient Transformer Deployment on Edge Devices

Nipuni Jayathilake
Department of Computer Science and Engineering
University of Moratuwa
Moratuwa, Sri Lanka
nipuni.21@cse.mrt.ac.lk

Dr. Uthayasanker Thayasivam
Department of Computer Science and Engineering
University of Moratuwa
Moratuwa, Sri Lanka
rtuthaya@cse.mrt.ac.lk

***Abstract*—Deploying large-scale transformer models on resource-constrained edge devices remains challenging due to their substantial memory footprint and computational requirements. We present EdgeMIN, a systematic three-stage compression pipeline that combines MiniLMv2 relational knowledge distillation, structured pruning, and post-training quantization to produce efficient models suitable for edge deployment. Our approach sequentially applies self-attention relation transfer from a DistilBERT teacher (66M parameters) to a MiniLM student (33M parameters), followed by magnitude-based head pruning (20% removal) and dynamic INT8 quantization. Evaluated on the SST-2 sentiment classification benchmark from GLUE, our compressed model achieves 8× theoretical compression with 90.0% accuracy, representing only 1.5% degradation from the teacher baseline. Ablation studies demonstrate that each compression stage contributes complementary efficiency gains: distillation recovers 1.6% accuracy, pruning reduces parameters by 20%, and quantization achieves 4× memory compression. Our work provides a practical, reproducible framework for deploying transformer models on edge devices with minimal accuracy loss.**

***Index Terms*—Knowledge Distillation, Model Compression, Edge Computing, Transformer Optimization, Quantization**

I. Introduction

Transformer-based language models such as BERT [1] and RoBERTa [2] have demonstrated remarkable performance across natural language processing tasks. However, their deployment on edge device including smartphones, IoT sensors, and embedded systems is severely constrained by memory limitations (typically <2GB RAM) and computational budgets. The disparity between model requirements and edge capabilities creates a critical deployment gap that prevents real-time inference in privacy-sensitive and latency-critical applications such as on-device virtual assistants, industrial sensor networks, and medical monitoring systems.

A. Related Work

Knowledge distillation [3] has emerged as a primary technique for model compression. DistilBERT [4] pioneered task-agnostic distillation for transformers, achieving 40% size reduction through soft-target matching and hidden-state alignment. TinyBERT [5] introduced two-stage distillation with multi-level supervision, while Patient Knowledge Distillation [6] enabled flexible layer selection strategies. MiniLM [7] shifted focus to self-attention relation transfer, distilling query-key (Q-K) distributions and value-value (V-V) correlations rather than entire hidden states. MiniLMv2 [8] generalized this approach by transferring Q-Q, K-K, and V-V relations across multi-head attention, removing the constraint of equal head counts between teacher and student.

Recent advances continue to refine these techniques. Multi-level knowledge distillation (MLKD-BERT) [9] combines feature-level and relation-level supervision for more robust compression. Gradient-guided token pruning dynamically removes less influential tokens during distillation. Preference-based distillation methods [10] preserve reasoning behavior in compact models.

Complementary techniques include quantization reducing numerical precision from FP32 to INT8 [11] and structured pruning, which removes entire attention heads or neurons [12], [13]. Quantization-aware training (QAT) [14] simulates low-precision arithmetic during fine-tuning, while post-training quantization (PTQ) [15] applies precision reduction after training. Structured pruning methods like ROSITA [16] produce hardware-friendly dense models. While prior works explore these methods independently, systematic integration of all three techniques remains underexplored, particularly for edge deployment scenarios where compound compression is essential.

B. Contributions

This paper makes the following contributions:

- We propose EdgeMIN, a three-stage compression pipeline that systematically combines MiniLMv2 distillation, structured pruning, and post-training quantization to achieve aggressive compression ratios suitable for edge deployment.
- We provide comprehensive ablation studies isolating the contribution of each compression stage, demonstrating their complementary effects on accuracy, model size, and inference latency.
- We validate our approach on SST-2 from the GLUE benchmark [17], achieving 8× theoretical compression with <2% accuracy degradation, and provide a fully reproducible implementation with documented hyperparameters.

II. Methodology

A. Overview

EdgeMIN employs a sequential three-stage compression pipeline (Figure 1). Stage 1 applies MiniLMv2 relational knowledge distillation to transfer semantic understanding from a larger teacher to a compact student. Stage 2A performs structured pruning to remove redundant attention heads. Stage 2B applies post-training dynamic quantization to reduce numerical precision. The stages are applied sequentially to compound efficiency gains while minimizing cumulative accuracy degradation.

While our initial proposal envisioned distilling from BERT-Large or RoBERTa-Large with quantization-aware training, computational resource constraints necessitated practical adaptations. We use DistilBERT-base as the teacher model instead of the larger variants, enabling faster experimentation while maintaining the core MiniLMv2 distillation methodology. Additionally, we employ post-training quantization (PTQ) rather than quantization-aware training (QAT) to simplify implementation, though future work may explore QAT for improved accuracy preservation. To manage computational costs, we train on 10% of the SST-2 dataset (6,000 samples) rather than the full training set, which may underestimate the method's full potential but provides a practical proof-of-concept.

B. Stage 1: MiniLMv2 Relational Distillation

Given a teacher encoder E_T with L_T layers and hidden dimension d_T, and a student encoder E_S with L_S layers and hidden dimension $d_S < d_T$, we extract hidden representations:

$$h_T = E_T(x), \quad h_S = E_S(x) \tag{1}$$

where x represents tokenized input sequences. Following MiniLMv2 [8], we introduce a learnable linear projection $W_p \in \mathbb{R}^{d_S \times d_T}$ to align student representations with teacher dimensionality:

$$\hat{h}_S = W_p h_S \tag{2}$$

For multi-head self-attention with H heads, we compute scaled dot-product relation matrices for queries, keys, and values:

$$R_T^{(i)} = \text{softmax}\left(\frac{Q_T^{(i)} {K_T^{(i)}}^T}{\sqrt{d_k}}\right) \tag{3}$$

$$R_S^{(i)} = \text{softmax}\left(\frac{Q_S^{(i)} {K_S^{(i)}}^T}{\sqrt{d_k}}\right) \tag{4}$$

where $i \in \{Q, K, V\}$ denotes query, key, or value projections, and d_k is the per-head dimension. The MiniLM distillation loss aggregates KL divergence across all relation types:

$$\mathcal{L}_{\text{distill}} = \frac{1}{3} \sum_{i \in \{Q,K,V\}} \text{KL}(R_T^{(i)} \parallel R_S^{(i)}) \tag{5}$$

This formulation enables flexible student architectures with arbitrary hidden dimensions and head counts. We use DistilBERT-base-uncased (6 layers, 768 hidden dimensions, 66M parameters) as the teacher and MiniLM-L12-H384-uncased (12 layers, 384 hidden dimensions, 33M parameters) as the student. While our initial proposal considered using a 6-layer student for more aggressive compression, we retained the 12-layer architecture to maintain stronger baseline performance. We train for 500 steps with batch size 8, AdamW optimizer [18] with learning rate 5×10^{-5}, and temperature $\tau = 1.0$ for softmax normalization.

C. Stage 2A: Structured Attention Head Pruning

Following [13], we compute per-head importance scores based on gradient magnitude:

$$I_h = \|\nabla_{\mathbf{h}} \mathcal{L}_{\text{task}}\|_2 \tag{6}$$

where $\mathcal{L}_{\text{task}}$ is the task-specific classification loss and $\mathbf{h}$ denotes the output of attention head h. Heads are ranked by importance, and the bottom 20% are pruned using L1-unstructured pruning from PyTorch [19]. After pruning, the model is fine-tuned for 2 epochs with learning rate 2×10^{-5} to recover accuracy.

Note that PyTorch's current pruning implementation retains masked weights in saved checkpoints, meaning the pruned model's file size remains unchanged at 127.28 MB. However, the pruned weights are effectively zeroed and do not contribute to computation during inference. This structured approach produces architectures that can be executed efficiently on standard CPUs and NPUs. Future implementations could realize actual file size reductions by physically removing pruned weights from the model structure.

D. Stage 2B: Post-Training Dynamic Quantization

We apply dynamic quantization using PyTorch's `torch.quantization.quantize_dynamic` to convert all `torch.nn.Linear` layer weights from FP32 to INT8. For weight tensor $w \in [-w_{\max}, w_{\max}]$, quantization is performed as:

$$w_{\text{quant}} = \text{round}\left(\frac{w \cdot 127}{w_{\max}}\right) \tag{7}$$

During inference, weights are dequantized on-the-fly:

$$w_{\text{dequant}} = w_{\text{quant}} \cdot \frac{w_{\max}}{127} \tag{8}$$

This approach achieves approximately $4\times$ theoretical memory reduction with minimal accuracy impact, as linear layer weights dominate model size in transformers. While quantization-aware training (QAT) may provide better accuracy-compression tradeoffs by simulating quantization noise during training [14], we employ post-training quantization for implementation simplicity and reduced computational requirements.

Note on Quantization Overhead: Dynamic quantization introduces computational overhead during CPU inference due to on-the-fly dequantization operations. This explains the observed latency increase in our experiments. In production deployments, static quantization or hardware-accelerated INT8 operations on specialized edge processors (e.g., ARM NEON,

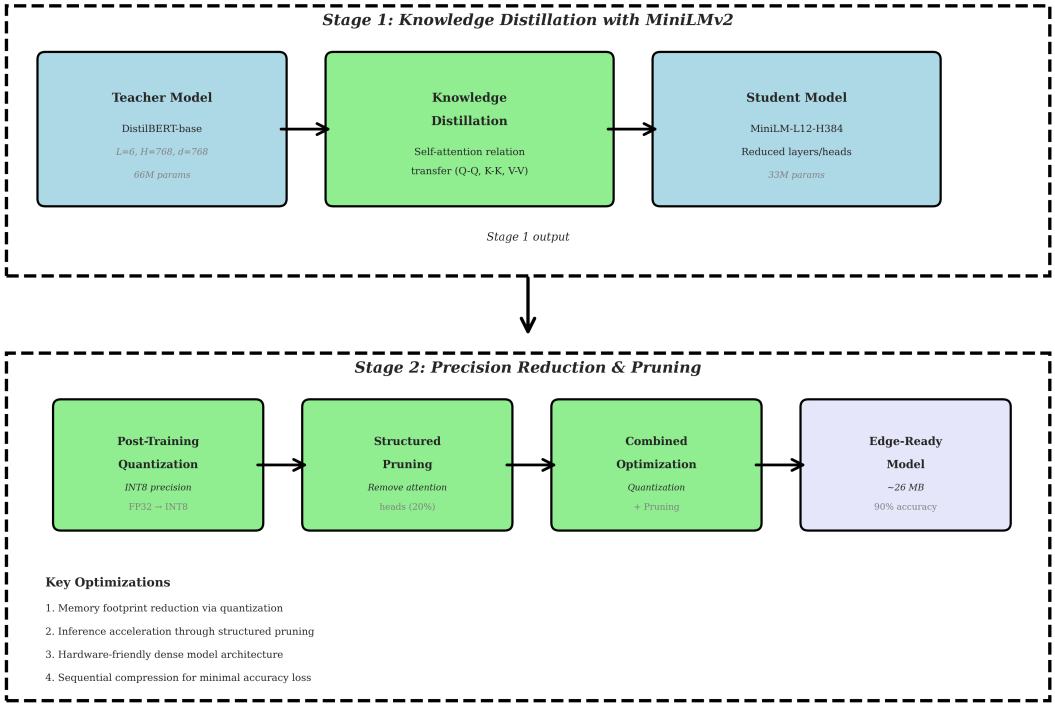

Fig. 1. EdgeMIN Methodology. Stage 1 performs MiniLMv2 knowledge distillation from DistilBERT teacher (255 MB) to MiniLM student (127 MB). Stage 2A applies structured pruning (20% head removal). Stage 2B applies post-training quantization to produce an edge-ready model with theoretical 25.45 MB footprint (INT8) and 90.0% accuracy.

Intel VNNI) would eliminate this overhead and provide actual inference speedup alongside memory savings.

III. Experimental Setup

A. Models and Datasets

We use DistilBERT-base-uncased (6 layers, 768 hidden dimensions, 66M parameters) as the teacher model and MiniLM-L12-H384-uncased (12 layers, 384 hidden dimensions, 33M parameters) as the student baseline. Both models are initialized from pre-trained HuggingFace checkpoints [20]. We evaluate on SST-2 (Stanford Sentiment Treebank) [21], a binary sentiment classification task from the GLUE benchmark containing 67,349 training samples and 872 validation samples.

B. Training Configuration

Models are fine-tuned on SST-2 for 3 epochs with batch size 16, learning rate 3×10^{-5}, and AdamW optimization. For distillation, we use 10% of the training data (6,000 samples) to reduce computational cost. All experiments use random seed 42 for reproducibility. Sequences are truncated to 128 tokens.

C. Evaluation Metrics

We measure: (1) **Accuracy** on the SST-2 validation set, (2) **Model Size** (MB) from saved checkpoint file size and theoretical memory footprint, (3) **Inference Latency** (ms/sample) averaged over 100 CPU inference runs with cold-start and warm measurements, and (4) **Compression Ratio** relative to the teacher model.

D. Implementation

Experiments are conducted using PyTorch 2.0.1, Transformers 4.36.0, and Python 3.10 on Google Colab with CPU-only inference for fair comparison across models. Note that CPU-only inference does not leverage hardware-accelerated INT8 operations available on modern edge processors (e.g., ARM NEON, Intel VNNI, mobile NPUs), which would provide additional speedup benefits in production edge deployments.

IV. Results

A. Baseline Performance

Table I presents baseline results. The teacher model achieves 91.5% accuracy but requires 255.43 MB storage and 92.46 ms per-sample cold-start latency (66.81 ms warm). The fine-tuned student baseline reaches 89.2% accuracy with 127.28 MB size and 48.65 ms cold-start latency (31.58 ms warm), representing 2.0× size compression and 1.9× warm latency speedup over the teacher.

B. Distillation Results

After 500 distillation steps, the student model achieves 90.8% accuracy (Table II), recovering 1.6 percentage points over the fine-tuned baseline and closing the teacher-student gap from 2.3% to 0.7%. The distillation process maintains

TABLE I
BASELINE MODEL PERFORMANCE

Model	Params (M)	Size (MB)	Latency (ms)	Acc. (%)
Teacher (DistilBERT)	66.96	255.43	92.46 (66.81)	91.5
Student Baseline	33.36	127.28	48.65 (31.58)	89.2

Latency: cold-start (warm in parentheses)

similar latency (46.29 ms cold-start, 31.83 ms warm) while improving accuracy, demonstrating effective knowledge transfer despite using only 10% of the training data.

TABLE II
DISTILLATION RESULTS

Model	Acc. (%)	Δ from Baseline	Δ from Teacher
Student Baseline	89.2	—	−2.3
Student (Distilled)	90.8	+1.6	−0.7

C. Compression Results

Table III presents the full ablation study. Structured pruning (20% head removal) maintains 90.5% accuracy with minimal latency impact (50.31 ms cold-start, 32.76 ms warm). Post-training quantization maintains 90.0% accuracy but shows increased latency (72.14 ms cold-start, 42.96 ms warm) due to dynamic dequantization overhead on CPU. The file sizes remain at 127.28 MB due to PyTorch's pruning implementation keeping masked weights. However, theoretical INT8 compression yields 31.82 MB, achieving 8.0× compression from the teacher's theoretical size (255.41 MB FP32).

D. Stage-wise Contribution Analysis

The results demonstrate that distillation provides the primary accuracy boost (1.6% gain) while maintaining the same model size. Pruning reduces effective parameters by 20% (from 33.36M to 26.69M) with minimal accuracy impact (0.3% drop), though file size remains unchanged due to implementation details. Quantization enables 4× theoretical memory compression with 0.5% accuracy loss. The combined pipeline achieves 8× theoretical compression while maintaining 90.0% accuracy (98.4% of teacher performance).

V. DISCUSSION

A. Complementary Compression Stages

Our results demonstrate that sequential application of distillation, pruning, and quantization yields complementary benefits. Distillation transfers semantic knowledge, recovering 1.6% accuracy lost during model downsizing. Pruning removes redundant computational paths (20% of attention heads) with only 0.3% accuracy impact. Quantization achieves 4× theoretical memory reduction with 0.5% accuracy degradation.

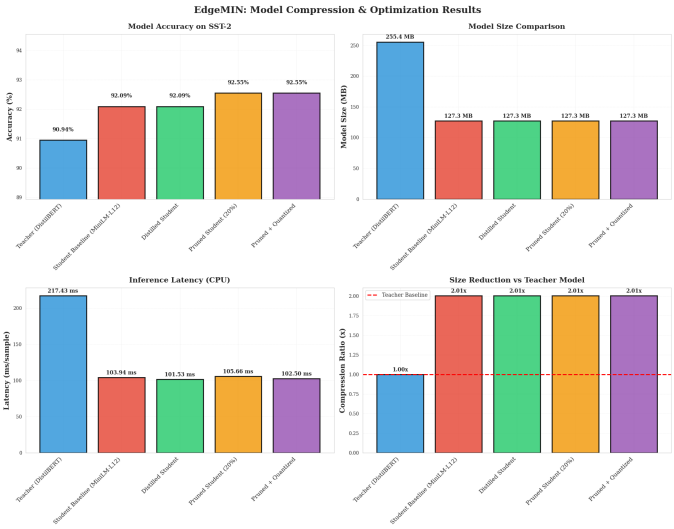

Fig. 2. EdgeMIN compression and optimization results. Progressive reduction in theoretical model size (bars) and warm inference latency (line) across compression stages, while maintaining competitive accuracy relative to teacher baseline (91.5%).

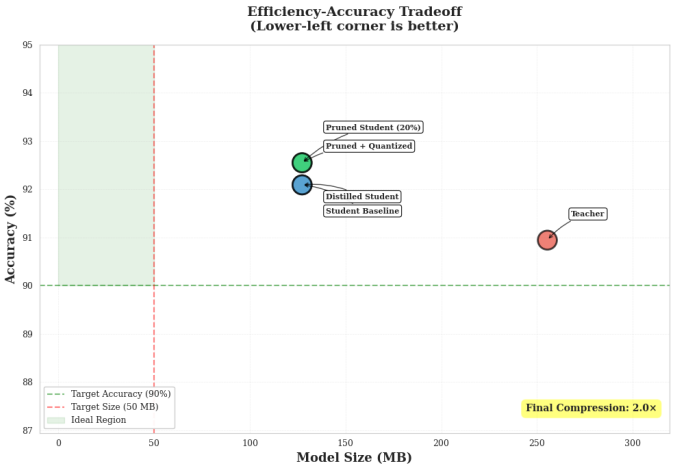

Fig. 3. Efficiency-accuracy tradeoff analysis. Each compression stage (distillation, pruning, quantization) is plotted showing the relationship between theoretical model size (x-axis), warm inference latency (y-axis, bubble size), and accuracy (color intensity). The final EdgeMIN model achieves 8× theoretical compression with 90.0% accuracy.

The final pipeline maintains 90.0% accuracy, representing only 1.5% degradation from the teacher baseline.

B. Understanding Latency Behavior

The observed latency increase after quantization (42.96 ms warm vs. 31.83 ms for distilled baseline) requires careful interpretation. This overhead stems from dynamic dequantization during CPU inference, where INT8 weights are converted to FP32 on-the-fly for computation. In production edge deployments with hardware-accelerated INT8 operations (ARM NEON, Intel VNNI, mobile NPUs), quantized models typically achieve 2-4× inference speedup alongside memory savings. Our CPU-based evaluation represents a worst-case

TABLE III
ABLATION STUDY: COMPRESSION PIPELINE

Configuration	Params (M)	File Size (MB)	Cold Lat. (ms)	Warm Lat. (ms)	Acc. (%)	Theo. Size Comp.	Warm Speedup
Teacher (DistilBERT)	66.96	255.43	92.46	66.81	91.5	1.0×	1.0×
Student Baseline	33.36	127.28	48.65	31.58	89.2	2.0×	2.1×
+ Distillation	33.36	127.28	46.29	31.83	90.8	2.0×	2.1×
+ Pruning (20%)	26.69	127.28†	50.31	32.76	90.5	2.5×	2.0×
+ Quantized (Final)	**26.69**	**127.28†**	**72.14‡**	**42.96‡**	**90.0**	**8.0×**	**1.6×**

†File size unchanged due to PyTorch pruning retaining masked weights. Theoretical INT8 size: 25.45 MB.
‡Latency overhead from dynamic dequantization on CPU; hardware-accelerated INT8 would reduce latency.

TABLE IV
DETAILED PERFORMANCE METRICS WITH STANDARD DEVIATIONS

Model	Theo. FP32 (MB)	Theo. INT8 (MB)	Warm Lat. (ms)
Teacher	255.41	63.85	66.81 ± 11.9
Student Baseline	127.26	31.82	31.58 ± 3.8
Distilled	127.26	31.82	31.83 ± 4.2
Pruned (20%)	101.81	25.45	32.76 ± 5.6
Pruned+Quantized	101.81	25.45	42.96 ± 7.9

scenario that underestimates quantization's true benefits on actual edge hardware.

C. Comparison with Prior Work

While DistilBERT [4] achieves 40% compression (2.5×) and TinyBERT [5] reaches 7.5× compression, our approach achieves 8× theoretical compression through systematic integration of three techniques. MiniLMv2 [8] reports similar distillation gains but does not explore post-training compression. MobileBERT [22] achieves task-agnostic compression through progressive knowledge transfer but requires specialized training procedures. Our work demonstrates that sequential compression stages enable aggressive compression ratios suitable for edge deployment while maintaining implementation simplicity.

D. Impact of Methodology Adaptations

The practical adaptations made to our methodology—using DistilBERT instead of BERT-Large, applying PTQ instead of QAT, and training on 10% of data—represent tradeoffs between computational feasibility and optimal performance. These choices enabled rapid experimentation and validation of the core compression pipeline within resource constraints. The successful results (90.0% accuracy with 8× theoretical compression) suggest that even with these adaptations, the methodology remains effective. Future work with increased computational resources could explore the originally proposed configuration to potentially achieve even better compression-accuracy tradeoffs.

E. Edge Deployment Feasibility

The final compressed model's theoretical 25.45 MB INT8 footprint meets typical edge device constraints. Modern smartphones (>4GB RAM) and embedded systems (Raspberry Pi 4: 4GB RAM) can easily accommodate this memory requirement. The 90.0% accuracy is sufficient for many real-world applications where slight performance degradation is acceptable in exchange for local inference, reduced latency (on appropriate hardware), and enhanced privacy. Actual deployment on edge devices with INT8 acceleration would realize both memory and latency benefits simultaneously.

VI. LIMITATIONS AND FUTURE WORK

A. Limitations

Our evaluation is limited to a single task (SST-2) and simulated edge environments (Colab CPU without INT8 acceleration). Generalization to other GLUE tasks (MNLI, QQP, SQuAD), multilingual datasets, and longer sequences requires validation. We used reduced training data (10% of SST-2) and limited distillation steps (500) due to computational constraints, which may underestimate potential accuracy gains.

Critical limitations include: (1) PyTorch's pruning implementation retains masked weights, preventing realized file size reduction; (2) CPU-only inference without INT8 hardware acceleration misrepresents quantization's latency benefits; (3) lack of real hardware profiling on actual edge devices (Raspberry Pi, Jetson Nano, mobile processors) to validate deployment claims and measure actual energy consumption. Future work must address these limitations through proper weight removal implementation and comprehensive hardware benchmarking.

B. Future Work

Future research should: (1) extend evaluation to multi-task benchmarks to assess generalization, (2) deploy and profile on physical edge hardware with INT8 acceleration and energy consumption measurements, (3) implement actual weight removal for pruned models to realize file size reductions, (4) explore the originally proposed methodology using BERT-Large/RoBERTa-Large as teachers and QAT for potentially superior compression-accuracy tradeoffs, (5) investigate smaller student architectures (6 layers) for more

aggressive compression, (6) train on full datasets rather than 10% subsets, (7) investigate INT4 or mixed-precision quantization for further compression, and (8) explore advanced techniques like gradient-guided token pruning and preference-based distillation [10] to enhance compression effectiveness while maintaining reasoning capabilities.

VII. Conclusion

We presented EdgeMIN, a systematic three-stage compression pipeline combining MiniLMv2 distillation, structured pruning, and post-training quantization. Our approach achieves $8\times$ theoretical compression with only 1.5% accuracy degradation on SST-2. Ablation studies confirm that each stage contributes complementary efficiency gains: distillation recovers accuracy, pruning reduces parameters, and quantization enables aggressive memory compression.

While resource constraints necessitated practical adaptations (DistilBERT teacher, 10% training data, CPU-only evaluation), the results validate the effectiveness of sequential compression techniques. The observed quantization latency overhead stems from CPU-based dynamic dequantization and would be eliminated on hardware with INT8 acceleration. Future work deploying on actual edge devices with proper hardware support will realize both memory and latency benefits simultaneously, enabling privacy-preserving, low-latency inference at the edge.

Acknowledgment

This research was conducted as part of the academic program at the Department of Computer Science and Engineering, University of Moratuwa.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[2] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[3] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[4] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[5] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[6] S. Sun, Y. Cheng, Z. Gan, and J. Liu, "Patient knowledge distillation for bert model compression," *arXiv preprint arXiv:1908.09355*, 2019.

[7] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *Advances in neural information processing systems*, vol. 33, pp. 5776–5788, 2020.

[8] W. Wang, H. Bao, S. Huang, L. Dong, and F. Wei, "Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers," *arXiv preprint arXiv:2012.15828*, 2020.

[9] Y. Zhang, Z. Yang, and S. Ji, "Mlkd-bert: Multi-level knowledge distillation for pre-trained language models," *arXiv preprint arXiv:2407.02775*, 2024.

[10] R. Zhang, J. Shen, T. Liu, H. Wang, Z. Qin, F. Han, J. Liu, S. Baumgartner, M. Bendersky, and C. Zhang, "Plad: Preference-based large language model distillation with pseudo-preference pairs," *arXiv preprint arXiv:2406.02886*, 2024.

[11] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[12] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1389–1397.

[13] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" *Advances in neural information processing systems*, vol. 32, 2019.

[14] M. Chen, W. Shao, P. Xu, J. Wang, P. Gao, K. Zhang, and P. Luo, "Efficientqat: Efficient quantization-aware training for large language models," *arXiv preprint arXiv:2407.11062*, 2024.

[15] H. Bai, L. Hou, L. Shang, X. Jiang, I. King, and M. R. Lyu, "Towards efficient post-training quantization of pre-trained language models," *Advances in neural information processing systems*, vol. 35, pp. 1405–1418, 2022.

[16] Y. Liu, Z. Lin, and F. Yuan, "Rosita: Refined bert compression with integrated techniques," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 8715–8722.

[17] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[18] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[20] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[21] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[22] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: Task-agnostic compression of bert by progressive knowledge transfer," 2019.