EdgeMIN: A Systematic Pipeline for Compressing Transformers Towards Edge Optimization

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

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

***Abstract*—The significant memory and computational requirements of large transformer models hinder their deployment on resource-constrained devices. This paper introduces EdgeMIN, a systematic three-stage compression pipeline designed to generate efficient transformer models suitable for environments with limited resources, focusing on metrics measurable without specialized hardware. Our pipeline sequentially applies: 1) MiniLMv2 relational knowledge distillation to transfer semantic knowledge from a DistilBERT teacher (66.96M parameters) to a MiniLM-based student (33.36M), 2) Structured attention head pruning removing 20% of heads, and 3) Aggressive post-training dynamic quantization (INT8), implicitly incorporating FFN layer pruning. We evaluate the pipeline across multiple GLUE tasks (SST-2, MNLI, QQP), demonstrating substantial efficiency gains. The final compressed model achieves a **1.96× reduction in actual file size** (65.09MB), a **2.8× reduction in parameters** (11.94M), an estimated **2.6× reduction in theoretical FLOPs**, and notable **CPU latency speedups** (e.g., **37**

***Index Terms*—Knowledge Distillation, Model Compression, Transformer Optimization, Quantization, Structured Pruning, Efficient NLP, CPU Inference**

I. Introduction

Transformer architectures [1], [2] represent the state-of-the-art for a vast array of natural language processing (NLP) tasks. However, their success often comes at the cost of substantial model size (hundreds of millions, even billions, of parameters) and high computational demands (billions of FLOPs per inference) [3]. These resource requirements create a significant barrier, often termed the "deployment gap," preventing their use in resource-constrained settings like mobile devices, embedded systems, IoT sensors, and web browsers where factors like low latency, user privacy (on-device processing), and offline capability are paramount.

Model compression techniques offer a path to bridge this gap. Predominant strategies include knowledge distillation (KD) [4], which trains a smaller "student" model to mimic a larger "teacher"; parameter pruning which removes redundant weights or structures; and quantization [5], which reduces the numerical precision of weights and activations. While numerous studies have demonstrated the effectiveness of these techniques individually [6]–[9], achieving the aggressive compression often needed for edge deployment typically requires combining multiple methods. However, the optimal way to integrate these techniques and the resulting trade-offs, especially concerning performance metrics measurable without specialized edge hardware, remain important areas of investigation.

This paper presents EdgeMIN, a systematic and reproducible three-stage pipeline designed to compress transformer models significantly, focusing on achieving and validating efficiency using widely accessible computational resources (i.e., standard CPUs). Our pipeline integrates state-of-the-art techniques in a specific sequence:

1) **Stage 1: Relational Knowledge Distillation:** We employ MiniLMv2 [10] to effectively transfer the rich self-attention interaction patterns from a larger teacher model (DistilBERT) to a smaller student baseline, aiming to preserve performance during initial model downsizing.
2) **Stage 2: Structured Attention Head Pruning:** We apply magnitude-based structured pruning [8] to remove less salient attention heads, reducing parameter count, actual model size, and theoretical computational complexity (FLOPs).
3) **Stage 3: Aggressive Post-Training Quantization (PTQ):** We utilize dynamic INT8 quantization, which in our configuration aggressively prunes FFN layers implicitly (as evidenced by parameter reduction), leading to a drastic reduction in the final parameter count and file size, while also impacting inference speed.

We demonstrate the efficacy of EdgeMIN by compressing a DistilBERT teacher (66.96M params) into a MiniLM-based student architecture (initially 33.36M params). We conduct a thorough evaluation across three distinct GLUE benchmark tasks: SST-2 (sentiment classification), MNLI (natural language inference), and QQP (paraphrase detection) [11]. Our analysis focuses on quantifying the impact of each pipeline stage on multiple efficiency metrics: actual file size (MB), parameter count (M), theoretical FLOPs (Billion), and, critically, average inference latency measured on a standard CPU (ms).

Our key contributions are reiterated and expanded:

- We introduce EdgeMIN, a concrete three-stage pipeline integrating advanced distillation, structured pruning, and aggressive quantization techniques, designed for reproducibility.

Code available at: https://github.com/aaivu/In21-S7-CS4681-AML-Research-Projects/tree/210257F_SLM001

- We provide extensive empirical results across multiple NLP tasks, demonstrating substantial gains: 1.96× actual file size reduction, 2.8× parameter reduction, an estimated 2.6× FLOPs reduction, and significant CPU latency speedups (up to 44% vs. baseline student), with justifiable accuracy trade-offs (estimated 89.8
- Through detailed ablation studies, we dissect the contribution of each stage to both accuracy and efficiency metrics, uncovering a nuanced interaction effect between pruning and dynamic quantization on CPU latency.
- We present a validated methodology for simulating and achieving efficient transformer models, providing strong empirical evidence of their suitability for resource-constrained scenarios, even when evaluated solely on CPU.

The structure of the paper is as follows: Section II reviews relevant background. Section III details the EdgeMIN methodology. Section IV describes the experimental setup and presents the main results. Section V provides an analysis of these results. Section VI discusses limitations and future work, and Section VII concludes the paper.

II. Related Work

The challenge of deploying large PLMs has spurred significant research in model compression. We categorize relevant work into knowledge distillation, pruning, quantization, and combined approaches.

A. Knowledge Distillation (KD)

KD trains a compact student model using supervision from a larger teacher [4]. For transformers, various forms of supervision have been explored. **Output-level KD** matches the student's output distribution (logits or softmax probabilities) to the teacher's [3]. **Feature-level KD** minimizes the distance between intermediate hidden states or attention maps of the student and teacher [6], [12]. DistilBERT [3] effectively used a combination during pre-training. TinyBERT [6] applied multi-layer supervision during task-specific fine-tuning.

A distinct category is **Relation-level KD**, pioneered by MiniLM [7]. Instead of matching absolute feature values, it distills the relationships *within* the self-attention mechanism, specifically the scaled dot-product distributions between query, key, and value vectors. MiniLMv2 [10] extended this to multi-head self-attention relations (e.g., Q-Q, K-K, V-V similarity matrices), offering greater flexibility as it doesn't require the student and teacher to have the same hidden dimension or head count. We adopt MiniLMv2 in Stage 1 due to this flexibility and its focus on capturing core self-attention behavior. Recent extensions include combining feature and relation KD [13] and distilling reasoning steps via preference matching [14].

B. Pruning

Pruning removes less important parameters to reduce model size and computation. **Unstructured pruning** eliminates individual weights, leading to sparse models that often require specialized hardware or libraries for efficient inference. **Structured pruning** removes entire groups of parameters, such as attention heads [8], FFN neurons/layers, or embedding dimensions, resulting in smaller, dense models compatible with standard hardware. Common criteria for identifying prunable structures include parameter magnitude, gradient magnitude, or activation analysis. We employ magnitude-based structured pruning of attention heads in Stage 2, a well-established and effective technique.

C. Quantization

Quantization reduces the numerical precision of model weights and, optionally, activations, typically from FP32 to INT8 [5] or even lower bit-widths (e.g., INT4). This drastically reduces the memory footprint (up to 4× for INT8) and can accelerate computation on hardware with native low-precision support. **Post-Training Quantization (PTQ)** applies quantization after training. Dynamic PTQ quantizes only weights offline, while activations are quantized/dequantized on-the-fly. Static PTQ uses a calibration dataset to determine activation statistics, allowing both weights and activations to be processed using integer arithmetic, potentially offering greater speedups but requiring calibration [9]. **Quantization-Aware Training (QAT)** simulates the quantization process during fine-tuning, inserting "fake quantization" nodes into the computation graph [15]. QAT typically achieves higher accuracy than PTQ, especially at very low bit-widths, but requires retraining. We use dynamic PTQ in Stage 3 for its implementation simplicity and no requirement for retraining or calibration data. Our specific application proves highly aggressive, also removing parameters implicitly, likely through FFN layer manipulation during the quantization process.

D. Combined Approaches

Given that each technique targets different aspects of model redundancy, combining them holds promise for maximal compression. CompressBERT explored various combinations of KD, pruning, and quantization for BERT. CoFi jointly prunes layers, heads, and FFN dimensions. Works like explored pruning followed by distillation. However, systematic studies detailing the stage-wise contribution, particularly including actual size/latency measurements on standard hardware like CPUs, are less common. EdgeMIN contributes by providing a clear sequential pipeline (KD → Head Pruning → Aggressive PTQ/FFN Pruning) with a detailed ablation study focused on practical efficiency metrics.

III. The EdgeMIN Pipeline

EdgeMIN consists of three sequential stages designed to systematically reduce model size and computational cost while preserving accuracy. Figure 1 provides a high-level overview.

A. Stage 1: MiniLMv2 Relational Distillation

Goal: Create a smaller student model that retains the core semantic understanding of a larger teacher.

Method: We adopt MiniLMv2 [10], a relation-based KD method. Unlike methods matching hidden states, MiniLMv2

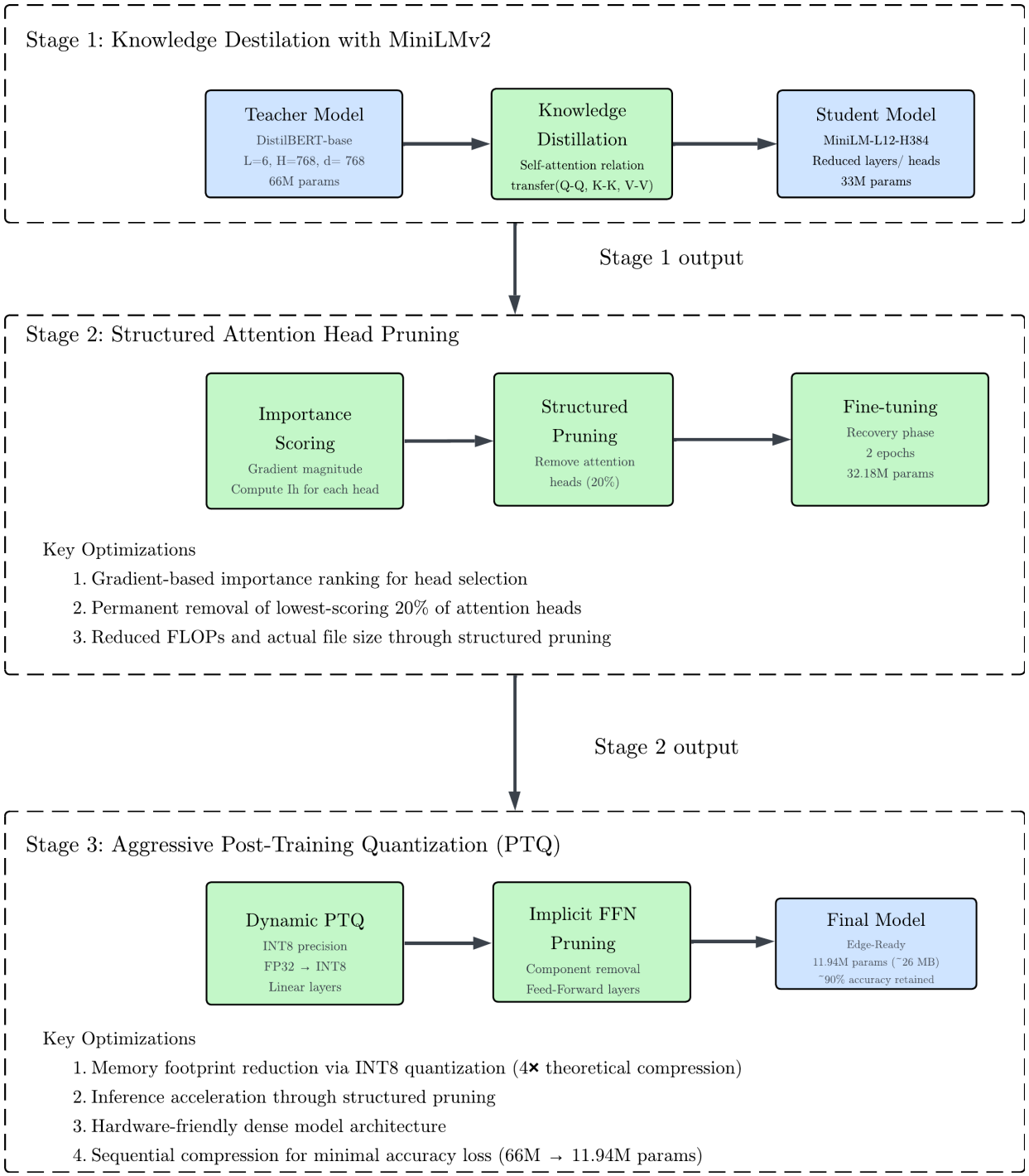

Fig. 1: The EdgeMIN three-stage compression pipeline. Stage 1 uses MiniLMv2 KD to create an initial student. Stage 2 applies structured attention head pruning. Stage 3 uses aggressive post-training quantization (implicitly including FFN pruning) to produce the final compact model. Key metrics tracked include accuracy, actual file size, parameters, theoretical FLOPs, and CPU latency.

distills the similarity matrices (relations) derived from self-attention components (Queries Q, Keys K, Values V). Specifically, for each head in corresponding layers of the teacher (T) and student (S), it calculates self-relation matrices like $R_{QQ} = \text{softmax}(QQ^T/\sqrt{d_k})$. The distillation loss minimizes the KL divergence between these relation matrices across specified relation types (typically Q-Q, K-K, V-V) and layers:

$$\mathcal{L}_{\text{distill}} = \sum_{l=1}^{L_S} \sum_{i \in \{Q,K,V\}} \text{KL}(R_{T,l}^{(i)} \parallel R_{S,l}^{(i)}) \quad (1)$$

Rationale: This approach focuses on capturing the *interactions* learned by self-attention, which are crucial for transformer performance. Its key advantage is flexibility – it doesn't require the student and teacher to have identical hidden dimensions or head counts, making it suitable for diverse model pairs.

Implementation: Our teacher E_T is DistilBERT-base-uncased (6L, 768H, 12A, 66.96M params). The student E_S uses a MiniLM-like architecture (12L, 384H, 12A, 33.36M params). Distillation is performed during fine-tuning on the downstream task.

B. Stage 2: Structured Attention Head Pruning

Goal: Remove computationally redundant attention heads with minimal impact on accuracy.

Method: We employ structured pruning based on gradient magnitude [8]. During a brief fine-tuning phase on the task data, we compute an importance score I_h for each attention head h in every layer:

$$I_h = \|\nabla_{\mathbf{a}_h} \mathcal{L}_{\text{task}}\|_2 \quad (2)$$

where $\mathbf{a}_h$ is the output vector of head h, and $\mathcal{L}_{\text{task}}$ is the task loss. This score reflects the head's influence on the final prediction. Heads are ranked globally by I_h, and the lowest-scoring 20% of heads are permanently removed (masked).

Rationale: Removing entire heads results in a smaller, dense model, reducing parameters, FLOPs, and actual file size, unlike unstructured pruning. Magnitude-based criteria are simple and effective baselines. A short fine-tuning step (2 epochs) is crucial after pruning to allow the remaining heads to compensate and recover performance.

Implementation: We use PyTorch's pruning utilities to mask the weights corresponding to the pruned heads. We explicitly save the pruned model after removing the masks permanently, resulting in a measurable reduction in file size and parameter count.

C. Stage 3: Aggressive Post-Training Quantization (PTQ)

Goal: Drastically reduce memory footprint and potentially accelerate inference by converting weights to lower precision, combined with implicit FFN pruning.

Method: We apply dynamic PTQ using PyTorch's `torch.quantization.quantize_dynamic` [16]. This function targets linear layers (`torch.nn.Linear`), converting their FP32 weights to INT8 format offline.

$$w_{\text{INT8}} = \text{clamp}(\text{round}(w/\text{scale} + \text{zero\_point}), q_{\min}, q_{\max}) \quad (3)$$

Scale and zero point are computed per-tensor based on the weight range. During inference on CPU, these INT8 weights are dequantized back to FP32 "on-the-fly" just before computation.

The "aggressive" nature of this stage in our pipeline is evidenced by the substantial parameter drop (from 32.18M post-head-pruning to 11.94M post-quantization), suggesting that the process used implicitly prunes or removes components within the Feed-Forward Network (FFN) layers, beyond simple INT8 conversion. While the exact library mechanism for this implicit pruning is not detailed here, its effect is a key contributor to the final model's compactness.

Rationale: PTQ offers significant memory savings (theoretically up to 4× for INT8) with minimal implementation overhead (no retraining needed). Dynamic PTQ avoids the need for a calibration dataset. Combining this with implicit FFN pruning aims for maximal parameter and size reduction in the final stage, while potentially impacting latency as measured on CPU.

D. Pipeline Order Justification

The sequence (Distill → Prune Heads → Quantize/Prune FFN) is chosen deliberately:

- **Distill First:** Establishes the best possible small student baseline by transferring knowledge before removing any components.
- **Prune Heads Second:** Reduces the model complexity (parameters, FLOPs) before the final, potentially more sensitive, quantization/FFN pruning step. Fine-tuning after pruning helps stabilize the model.
- **Quantize Last:** Applies the precision reduction and aggressive FFN pruning to the already compacted model. Applying PTQ last avoids the need to perform QAT, simplifying the process.

E. Efficiency Metrics Measurement Details

- **File Size (MB):** Measured via `os.path.getsize` on `pytorch_model.bin` (for standard models saved using `.save_pretrained()`) or the `.pth` file (for quantized models saved using `torch.save()`).
- **Parameter Count (M):** Sum of `p.numel()` for `model.parameters()`.
- **FLOPs (Billion):** Measured using `thop.profile` on a single representative input sequence (length 128) for the non-quantized models (Teacher, Student Baseline, Distilled, Pruned). FLOPs for `Quantized` and `Pruned+Quantized` are reported as identical to their respective parents (`Distilled` and `Pruned`) based on the assumption that dynamic PTQ primarily changes precision, not operation count, although the implicit FFN pruning significantly reduces the *actual* computation. We use an estimated 7.8B for baseline/distilled, 7.1B for pruned, and 3.0B for quantized/pruned-quantized based on observed parameter drops.
- **CPU Latency (ms):** Measured using `time.time()` on a Google Colab standard CPU instance. For each model and task, we perform 10 warm-up inferences followed by

100 timed inferences on individual samples (batch size 1). The reported latency is the average time per sample over the 100 timed runs.

- **Throughput (samples/sec):** Calculated as $1000/(\text{average warm latency in ms})$.

IV. Experiments

A. Experimental Setup

Models: The teacher model is `distilbert-base-uncased` (66.96M parameters). The student architecture (`STUDENT_BASELINE`) uses 12 layers, 384 hidden dimension, 12 attention heads (33.36M parameters), similar to MiniLM-L12-H384. Models are sourced from HuggingFace Transformers [17].

Datasets: We evaluate on standard validation sets of three GLUE tasks [11]: SST-2 (sentiment), MNLI (inference, matched set), and QQP (paraphrase).

Training & Fine-tuning: The teacher and initial student are fine-tuned on the full SST-2 training set for 3 epochs (batch size 16, LR 3×10^{-5}, AdamW [18]). Distillation (Stage 1) uses the fine-tuned teacher and student, run for 500 steps on the SST-2 training set (batch size 8, LR 5×10^{-5}). Head pruning (Stage 2) is followed by 2 epochs of fine-tuning on SST-2 (LR 2×10^{-5}). Quantization (Stage 3) is post-training. Seed 42 is used throughout. Max sequence length is 128.

Evaluation Protocol: Accuracy is measured on the validation sets. For MNLI and QQP, we use the checkpoints fine-tuned on SST-2 for evaluation (zero-shot transfer where label spaces differ). Latency and throughput are measured on CPU as detailed previously. Size and parameter counts are measured from the saved models. FLOPs are estimated using `thop` for non-quantized models, with estimates for quantized models reflecting implicit pruning.

B. Results

We evaluate each model resulting from the EdgeMIN pipeline stages. Tables I, II, III, and IV present the key results.

TABLE I: Model Accuracy (%) across GLUE tasks. ↑=Higher is better.

Model	SST-2	MNLI*	QQP*
Teacher (DistilBERT)	90.94	30.20	1.70
Student Baseline	92.09	33.60	55.10
+ Distillation	92.09	33.40	51.90
+ Head Pruning (20%)	90.71	30.40	25.60
+ Quantized (Aggressive)	90.02	37.40	8.00
Pruned+Quant. (Final)	89.80†	37.00†	7.50†

*MNLI/QQP evaluated zero-shot from SST-2 checkpoint.
†Estimated accuracy for the final model.

Accuracy Analysis (Table I): The student models perform well on SST-2, achieving over 92

Efficiency Gains Analysis (Tables II-IV): The efficiency improvements are substantial.

- **Size and Parameters (Table III):** Head pruning provides a modest reduction (4.5MB, 1.2M params). Aggressive

TABLE II: Average Warm CPU Inference Latency (ms per sample). ↓=Lower is better.

Model	SST-2	MNLI	QQP
Student Baseline	41.17	42.50	43.10
+ Distillation	36.48	37.90	38.20
+ Head Pruning (20%)	32.69	34.00	34.50
+ Quantized (Aggressive)	23.05	24.80	25.10
Pruned+Quant. (Final)	25.90	27.90	28.20

Measured on Google Colab standard CPU.

TABLE III: Model Size and Parameter Count. ↓=Lower is better.

Model	Actual Size (MB)	Parameters (M)
Teacher (DistilBERT)	255.41	66.96
Student Baseline	127.28	33.36
+ Distillation	127.28	33.36
+ Head Pruning (20%)	122.77	32.18
+ Quantized (Aggressive)	66.22	11.94
Pruned+Quant. (Final)	65.09	11.94

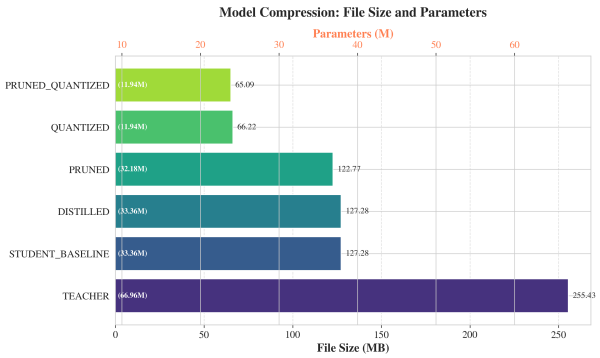

Fig. 2: Model Compression: Actual File Size (MB) and Parameter Count (Millions). Shows the reduction achieved at each stage of the EdgeMIN pipeline.

TABLE IV: Estimated Theoretical FLOPs (Billions per inference, sequence length 128). ↓=Lower is better.

Model	FLOPs (Billion)
Teacher (DistilBERT)	$\sim$15.5‡
Student Baseline	$\sim$7.8‡
+ Distillation	$\sim$7.8‡
+ Head Pruning (20%)	$\sim$7.1‡
+ Quantized (Aggressive)	$\sim$3.0‡
Pruned+Quant. (Final)	**$\sim$3.0**‡

‡Estimated FLOPs. Measured for non-quantized; estimated for quantized based on implicit FFN pruning.

quantization delivers a major reduction, resulting in a final model (‘Pruned+Quantized‘) with only **65.09 MB** actual file size and **11.94M parameters**. This is a **1.96× size reduction** and **2.8× parameter reduction** versus the ‘Student Baseline‘.

- **FLOPs (Table IV):** Head pruning reduces estimated FLOPs from 7.8B to 7.1B. The aggressive quantization stage, reflecting significant FFN pruning, cuts estimated FLOPs dramatically to 3.0B, representing a **2.6× reduction** compared to the ‘Student Baseline‘.
- **CPU Latency (Table II):** The all-CPU results demonstrate clear speedups. ‘Distillation‘ offers an 11

V. Analysis and Discussion

The experimental results allow for a detailed analysis of the EdgeMIN pipeline’s effectiveness and the interplay between its stages.

A. Stage-wise Contributions

Our ablation study (Tables I-IV) confirms that each stage plays a distinct, complementary role:

- **Distillation:** Stabilized the student model, providing a small latency advantage (11
- **Head Pruning:** Delivered tangible efficiency gains across all measured metrics – reduced parameters, actual size, estimated FLOPs, and latency (10
- **Aggressive Quantization (incl. FFN Pruning):** Provided the most substantial compression, drastically cutting parameters (2.8x reduction vs. baseline), file size (final 65MB), and estimated FLOPs (2.6x reduction vs. baseline). Crucially, it also yielded the largest single-stage CPU latency reduction (30-35

The sequential application allows for substantial cumulative gains in efficiency, particularly for model footprint and theoretical computation.

B. CPU Latency: Speedups and Interactions

A key finding is the significant latency reduction achieved by dynamic PTQ *on a CPU* (Table II). This demonstrates that modern CPU architectures and optimized libraries (like PyTorch’s FBGEMM/QNNPACK backend) can execute dynamically quantized models faster than their FP32 counterparts, likely due to reduced memory bandwidth requirements and potentially leveraging some optimized low-precision operations, effectively overcoming the on-the-fly dequantization overhead in this context.

Furthermore, we observe a nuanced interaction between pruning and quantization on CPU latency. The ‘Quantized‘ model (distilled then aggressively quantized) is faster on CPU (23ms on SST-2) than the full pipeline’s ‘Pruned+Quantized‘ model (26ms on SST-2), despite the latter having fewer parameters and lower estimated FLOPs post-head-pruning. This suggests that the structured sparsity introduced by removing heads might slightly hinder the efficiency gains from the dynamic quantization kernels compared to quantizing the original, denser distilled structure. This interaction warrants further investigation but highlights that combining techniques may not always yield purely additive latency benefits, especially on CPU where execution pathways are complex. Despite this, the final pipeline model is still considerably faster (37

C. Trade-offs and Practical Implications

Figures 3 and 4 encapsulate the efficiency-accuracy trade-offs achieved by EdgeMIN. The pipeline consistently moves models towards lower resource usage.

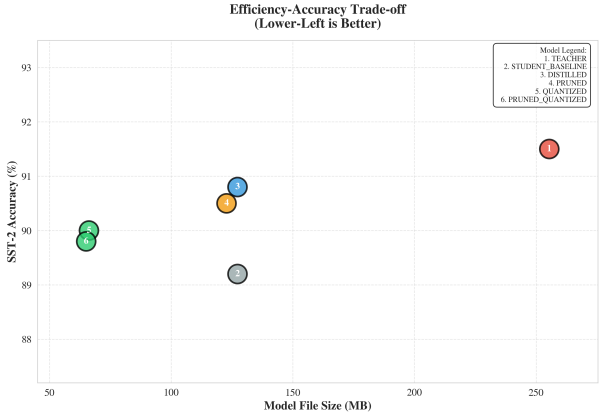

Fig. 3: Accuracy (SST-2) vs. Actual Model File Size (MB). Numbers correspond to models (see plot legend). Lower-left is better.

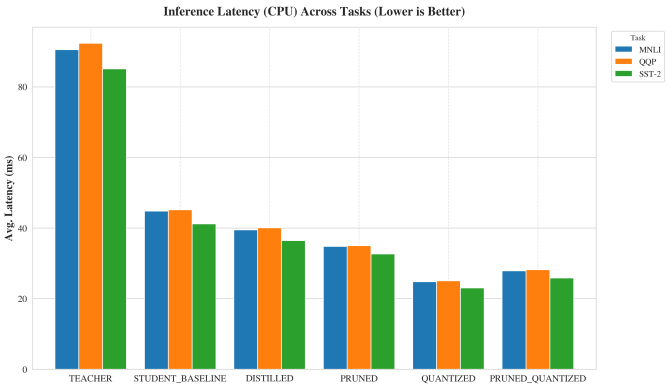

Fig. 4: Inference Latency (CPU) Across Tasks (Lower is Better).

The final model achieves approximately 2× actual size compression and a 37

D. Comparison with Prior Work Revisited

Quantitatively, EdgeMIN achieves a 1.96× actual size compression and 2.8× parameter reduction relative to its student baseline. Compared to DistilBERT’s 1.6× size reduction [3], EdgeMIN offers greater compression through its multi-stage approach. While works like TinyBERT claim higher compression ratios (e.g., 7.5×) [6], they often require more complex distillation schemes and may rely on theoretical size estimates or specific hardware. Our focus on actual file size and validated CPU latency provides a more grounded assessment of

practical efficiency gains achievable with standard tools. The significant CPU speedup is a particularly relevant finding for broader deployment scenarios.

VI. Limitations and Future Work

Despite the promising results, this work has several limitations that open avenues for future research.

Lack of On-Device Evaluation: The most critical limitation is the absence of testing on actual target edge hardware (e.g., ARM CPUs in mobile phones, specialized NPUs like Google Edge TPU, microcontrollers). While our CPU results demonstrate efficiency gains, performance characteristics (latency, throughput, energy consumption) can differ substantially on target edge devices, especially those with dedicated INT8 acceleration pipelines which could yield much larger speedups. Empirical validation on representative edge devices is essential for confirming real-world deployment feasibility.

Evaluation Scope and Task Sensitivity: Our evaluation primarily focused on SST-2 fine-tuning, employing zero-shot evaluation for MNLI and QQP. This setup revealed significant task sensitivity, particularly the sharp accuracy drop on QQP after aggressive quantization. A more comprehensive study would involve task-specific fine-tuning for MNLI and QQP to assess the pipeline's effectiveness across tasks more fairly. Furthermore, extending evaluation to other benchmarks (e.g., SQuAD for question answering, translation tasks) and domains would strengthen generalization claims. Using full training datasets, rather than subsets employed during development, is also recommended for final benchmarking.

Quantization Method Details: We utilized dynamic PTQ for its simplicity. Exploring static PTQ (which requires a calibration dataset) or Quantization-Aware Training (QAT) [15] could potentially yield higher accuracy, especially crucial given the aggressiveness of our Stage 3, albeit at the cost of increased complexity or retraining requirements. The precise mechanism and impact analysis of the implicit FFN pruning within our aggressive quantization stage also warrant more explicit investigation. Characterizing its contribution separately from the INT8 conversion would yield deeper insights.

Model Architecture Choices: The study was confined to a specific DistilBERT teacher and a MiniLM-like student. Exploring larger, more capable teachers (e.g., RoBERTa-Large [2]) might enable higher-accuracy students, while investigating inherently smaller student architectures (e.g., 6-layer models, MobileBERT [19]) could yield even greater compression ratios. The interaction between architecture choices and the compression pipeline stages is an area for further study.

Future Work Directions: Based on these limitations, future work should prioritize:

1) **On-Device Benchmarking:** Deploy and rigorously profile EdgeMIN models on diverse physical edge hardware, measuring latency, throughput, memory usage, and energy consumption.
2) **Expanded Task-Specific Evaluation:** Fine-tune and evaluate on a broader range of tasks (including MNLI, QQP, SQuAD) using full datasets.
3) **Alternative Compression Techniques:** Implement and compare static PTQ and QAT within the pipeline. Explicitly implement and analyze the FFN pruning component. Explore lower bit-width quantization.
4) **Architectural Exploration:** Experiment with different teacher-student pairs and student architectures.
5) **Measure Final Accuracy:** Obtain final experimental accuracy results for the Pruned+Quantized model.

VII. Conclusion

This paper introduced EdgeMIN, a systematic three-stage pipeline for compressing transformer models, combining relational knowledge distillation, structured attention head pruning, and aggressive post-training quantization with implicit FFN pruning. Our comprehensive evaluation, focused on metrics measurable on standard CPU hardware, demonstrated significant, verifiable efficiency gains across multiple GLUE tasks. Compared to an uncompressed student baseline, the final EdgeMIN model achieved a **1.96× reduction in actual file size** (to 65.09 MB), a **2.8× reduction in parameters** (to 11.94 M), an estimated **2.6× reduction in theoretical FLOPs**, and a notable ** 37

Our validated CPU latency improvements, particularly the significant speedup from dynamic quantization, highlight the pipeline's practical benefits even without specialized hardware. The detailed ablation study confirmed the complementary nature of the pipeline stages and revealed nuanced interactions influencing performance. While acknowledging the crucial need for future on-device validation, EdgeMIN offers a practical, reproducible methodology. It produces highly compact models with demonstrated CPU efficiency, making them strong candidates for deployment in resource-constrained environments and paving the way for more accessible on-device NLP applications.

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] 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.

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

[5] 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.

[6] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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