Expert Attention: Structured Head Specialization and Pruning for Fast Transformer Inference

Anonymous ACL submission

Abstract

Encoder-only Transformers excel at language understanding but still suffer from inefficiency in **large-scale inference and annotation scenarios**, as well as in **compute-constrained deployments**, due to redundant attention heads and heavy feed-forward layers. We propose *Expert Attention*, a structured pruning pipeline that reinterprets each attention head as an independent expert paired with a lightweight shared Expander FFN. Training adopts a **layer-wise progressive strategy** with a **two-stage schedule**—balanced exploration followed by specialization—to ensure stability and expert differentiation. Our **Top-1 routing–pruning design** is used *during training* to select a single expert per layer and collect clear usage statistics; *post-training*, we perform usage-driven Top-1 pruning, remove the router entirely, and retain only the selected experts, yielding **deterministic, router-free inference** without any dynamic activation. Even with up to 11/12 layers pruned, Expert Attention maintains over 88% accuracy while reducing parameters by 88.5% and boosting throughput by 4.2× on GLUE. The method is simple, hardware-friendly, and deployment-ready, providing a practical solution for efficient Transformer inference.

1 Introduction

In recent years, Transformer-based encoder models such as BERT (Devlin et al., 2019) and XLM-R (Conneau et al., 2020) have achieved remarkable progress in language understanding tasks (Wang et al., 2018, 2020). Benefiting from their bidirectional attention, encoder models offer strong expressive power and full parallelism over the input during inference, making them the superior choice for classification, tagging, and other text understanding tasks compared with much larger decoders (Yang et al., 2024; DeepSeek-AI et al., 2025; Touvron et al., 2023).

Nonetheless, even these encoder models suffer from inefficiencies in large-scale deployments - such as multilingual analysis over massive social media streams - where inference speed and model size become critical bottlenecks. To address this, numerous pruning and compression techniques have been proposed. They fall broadly into two categories:

- **Coarse-grained pruning**, such as layer dropping and attention head pruning (Michel et al., 2019; Voita et al., 2019), structurally simplifies the architecture but often degrades task performance.
- **Fine-grained pruning**, including structured sparsity and weight magnitude pruning (Xu et al., 2020; Sanh et al., 2020), can yield high compression rates, but typically requires specialized tooling adaptation and introduces complex deployment challenges.

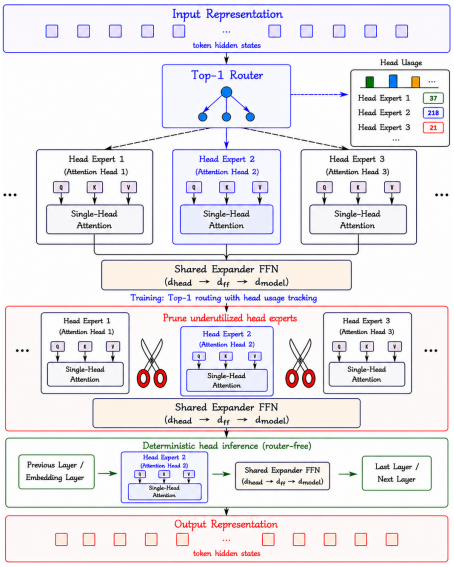

Figure 1: Overview of **Expert Attention**. During training, Top-1 routing selects a single attention-head expert and tracks expert usage statistics. After training, underutilized experts (marked with scissors) are pruned, and the routing module is removed to obtain compact router-free deterministic inference.

Separately, Mixture-of-Experts (MoE) models (Shazeer et al., 2017; Lepikhin et al., 2020) have shown promise in scaling model capacity without proportional computational cost. Recent works such as MoH (Jin et al., 2025) extend this idea by reinterpreting multi-head attention as a mixture of experts, allowing each token to dynamically select attention heads. While effective in boosting performance, these approaches are often architecture-specific and do not directly address inference efficiency or model compression.

In this work, we introduce **Expert Attention**—not as a new attention mechanism—but as a practical and structured pruning pipeline for pretrained Transformers. Our goal is to reframe each attention head as a candidate expert that can be specialized, evaluated, and pruned. As illustrated in Figure 1, each attention head is decoupled into an independent expert and paired with a lightweight shared Expander FFN. During training, a lightweight **Top-1 routing mechanism** selects a single expert per layer, enforcing sparsity and providing clear usage statistics. After training, we apply usage-driven **Top-1 pruning**, removing underutilized experts and discarding the router entirely. This Top-1 routing–pruning design yields a compact, deterministic encoder without dynamic routing at inference, while maintaining high task performance.

Our contributions are as follows:

- We propose **Expert Attention**, a lightweight pipeline that treats each attention head as a structurally independent expert, paired with a simplified FFN module to preserve functional capacity after pruning.
- We propose a **layer-wise progressive strategy with a two-stage schedule**—first balanced exploration and then specialization—to guide expert head usage without auxiliary losses or supervision signals.
- We apply a **Top-1 usage-driven pruning strategy**, removing underused heads entirely post-training, and converting dynamic routing into static deterministic inference.
- Our method is **purely architectural**, requiring no custom operations or scoring functions, and significantly improves inference throughput with minimal accuracy loss - outperforming prior head pruning techniques.

2 Related Work

Attention Head Pruning Prior studies have shown that many attention heads in Transformer models are redundant. Michel et al. (2019) demonstrated that large fractions of heads can be pruned without significant loss in accuracy. Voita et al. (2019) introduced auxiliary losses to identify useful heads, while others proposed learnable masking or score-based heuristics (Michel et al., 2019; Li et al., 2021). While effective, these methods often require custom scoring functions or supervised signals, and typically focus only on the attention sublayer.

Transformer Compression Techniques Transformer compression has been widely explored to improve deployment efficiency. Coarse-grained methods like LayerDrop (Fan et al., 2019) and DynaBERT (Hou et al., 2020) prune layers or adapt model depth/width dynamically. Fine-grained methods such as Movement Pruning (Sanh et al., 2020) and quantization techniques (Zafrir et al., 2019) operate at the parameter level but often introduce additional complexity and may require custom inference frameworks. Few of these methods offer structured, architecture-level pruning that preserves model modularity.

Mixture-of-Experts (MoE) for Expansion MoE architectures (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022) scale model capacity via conditional computation, typically by routing tokens to subsets of FFN experts. These approaches enable large-scale training without proportional increases in compute, but are designed primarily for expansion, not compression. Moreover, most MoE methods target FFN blocks, leaving attention layers structurally untouched.

MoE-Inspired Attention Head Routing Recent work has explored applying MoE principles to attention heads. Notably, MoH (Jin et al., 2025) reformulates multi-head attention as a dynamic mixture of heads, introducing a routing mechanism that selectively activates heads per token. MoH focuses on improving performance in vision and LLM contexts, and does not target structured pruning or inference efficiency. In contrast, our method reinterprets head modularity as a means for compression, not for architectural replacement.

Our Contribution We propose a structured pruning pipeline—Expert Attention—that treats each

attention head as an independent expert and enables post-training pruning via usage statistics. Unlike prior head pruning approaches, we simplify head structure, pair each with a lightweight FFN, and adopt a two-stage training strategy that naturally encourages specialization. Unlike MoE-based expansion works, our approach requires no dynamic routing at inference and directly targets throughput and parameter reduction in pretrained encoder models.

3 Method

3.1 Expert Attention

Expert Attention is a structured pruning pipeline designed to improve inference efficiency in Transformer encoders. Instead of modifying the attention mechanism itself, we reorganize each attention head into an independent expert path that can be selectively activated and later pruned.

The standard Transformer computes all heads in parallel and applies a large shared FFN afterward. Directly pruning heads in this setup often degrades performance. To avoid this, we restructure first—isolating heads and pairing them with lightweight feed-forward blocks—then prune based on usage after training.

Our design introduces three key components:

- **Expert modularization**: Each head is decoupled and paired with a shared, low-cost Expander FFN, forming a self-contained expert.
- **Top-1 routing**: A simple gating mechanism activates only the most relevant expert per input, enforcing sparsity and enabling usage tracking.
- **Two-stage training**: The model first explores diverse head usage, then specializes experts for pruning without auxiliary losses.

As shown in Figure 1, Expert Attention modularizes attention heads into standalone experts, each paired with a shared Expander FFN. During training, Top-1 gating guides expert usage, and after training, unused experts (scissors) are pruned, enabling aggressive compression without structural inconsistency.

In the next subsections, we describe the expert formulation, training process, and pruning strategy that comprise the Expert Attention pipeline.

3.1.1 Attention as Expert Module

In Expert Attention, we reinterpret each attention head as an independent expert module. Unlike standard multi-head attention, where heads are computed jointly and fused, our design detaches each head into a self-contained unit with its own query, key, and value projections:

$$Q_i = XW_i^Q, \quad K_i = XW_i^K, \quad V_i = XW_i^V$$

Each expert computes scaled dot-product attention individually:

$$\text{Head}_i(X) = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right) V_i$$

These heads are not concatenated or merged. Instead, they act as standalone expert branches that will be selectively routed during inference.

By treating each head independently, we allow each to specialize without interference from others. This modularization also lays the foundation for subsequent pruning, as unused heads can be easily removed.

3.1.2 Expander FFN

In standard Transformer layers, the feed-forward network (FFN) is a two-layer fully connected block applied after the multi-head attention output, projecting from the hidden dimension d to an intermediate dimension and back to d. This design introduces a significant number of parameters and dominates the overall model size.

In Expert Attention, we replace the original FFN with a simplified and shared **Expander FFN**, which serves three purposes:

1. **Dimensional adaptation:** Since each attention expert produces an output of shape (B, L, d_{head}), where $d_{\text{head}} = d/h$, we need to project this lower-dimensional output back to the full hidden dimension d. This projection ensures compatibility with the residual connection and layer normalization applied afterward.
2. **Functional substitution:** The Expander FFN also replaces the original FFN, providing the necessary non-linear transformation and capacity to support downstream representation learning.
3. **Parameter efficiency:** Unlike the original FFN, which maps $d \to d_{\text{ff}} \to d$, our Expander

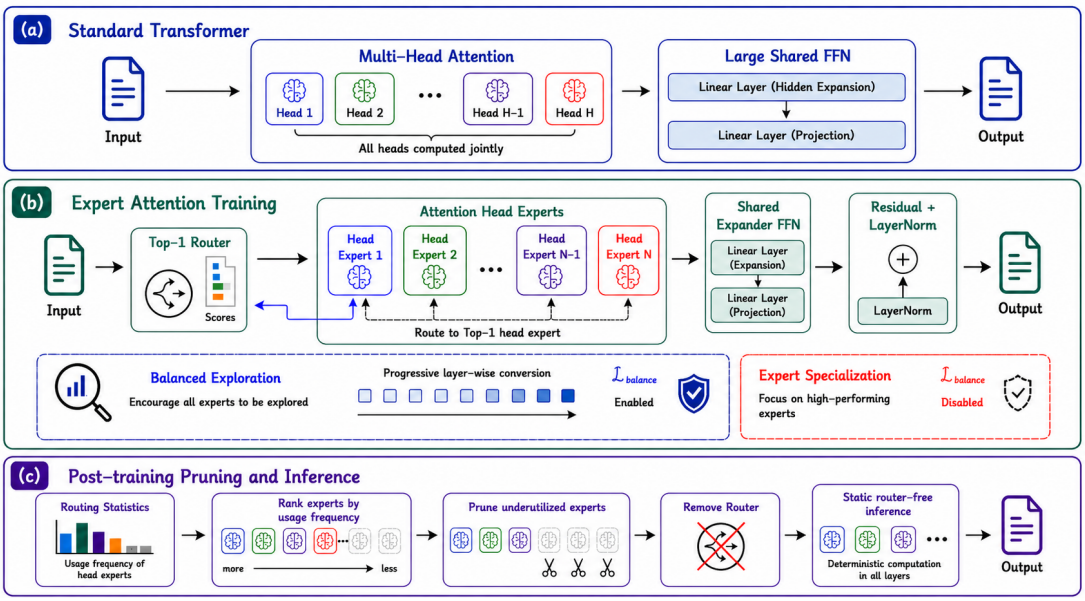

Figure 2: Full pipeline of **Expert Attention**. (a) Standard Transformer jointly computes all attention heads followed by a large shared FFN. (b) Expert Attention decouples attention heads into independent experts with sparse Top-1 routing and a shared lightweight Expander FFN. Training adopts a two-stage strategy consisting of balanced exploration and expert specialization together with progressive layer-wise conversion. (c) After training, experts are ranked by routing frequency and infrequently selected experts are pruned, after which the routing module is removed to obtain compact and efficient router-free deterministic inference.

FFN only maps $d_{\text{head}} \to d_{\text{ff}} \to d$. This greatly reduces the number of parameters, especially when $d_{\text{head}} \ll d$.

Formally, the Expander FFN is defined as:

$$\text{ExpFFN}(x) = W^2 \cdot \text{GELU}(W^1 x)$$

where $W^1 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{head}}}$ and $W^2 \in \mathbb{R}^{d \times d_{\text{ff}}}$. All experts share the same Expander FFN parameters, which promotes efficiency while maintaining model capacity through sparse expert selection.

As illustrated in Figure 2, unlike the standard Transformer where all heads are aggregated and passed to a large FFN, Expert Attention first decouples heads into experts and then employs a shared lightweight Expander FFN. This block expands each low-dimensional head output back to the full hidden dimension while adding non-linear transformation, thereby replacing the original FFN with a more parameter-efficient alternative.

3.1.3 Top-1 Gating

We adopt a hard routing strategy that selects only a single expert per input ($k = 1$). This design choice is driven purely by empirical observation: during pruning, we found that using multiple experts ($k > 1$) leads to significant accuracy degradation, while Top-1 routing consistently preserves performance.

The gating mechanism projects the [CLS] token representation $x_{\text{CLS}} \in \mathbb{R}^d$ to a score vector over N experts:

$$g = \text{Linear}(x_{\text{CLS}}) \in \mathbb{R}^N$$

Then the top expert is selected as:

$$i^* = \arg\max_j g_j$$

Only the selected expert i^* is executed; no softmax or weighting is applied. This simple routing not only reduces computation but also produces clear usage patterns, making expert pruning straightforward.

We validate the impact of this choice in Section 4.3, where we compare Top-k routing variants under different pruning levels.

3.1.4 Integration and Output

As illustrated in Figure 2, once the top-1 expert i^* is selected, its attention head processes the input and passes the result through the shared Expander FFN:

$$E = \text{ExpFFN}(\text{Head}_{i^*}(X))$$

We then apply residual connection and layer normalization, using the original input X:

$$Y = \text{LayerNorm}(E + X)$$

This design ensures that only a single expert is executed per layer and per input. Because experts are fully decoupled, those that are rarely or never selected can be pruned entirely—enabling efficient post-training compression without any structural reconfiguration.

3.2 Further training

Training an Expert Attention model poses unique challenges due to the sparse and modular nature of its computation. In particular, directly applying MoE transformation to all layers at once often leads to instability or training collapse, especially when fine-tuning on downstream tasks. To address this, we adopt a progressive training strategy with two critical components: **layer-wise MoE conversion** and a **two-stage optimization schedule**.

Layer-wise MoE Conversion. Instead of transforming all encoder layers into Expert Attention modules simultaneously, we convert one layer at a time in a staged manner. At each training epoch, one additional Transformer layer is replaced by its MoE counterpart. Once all target layers have been converted, we continue training the full model for a small number of extra epochs (typically 2) to stabilize learning and allow for full adaptation.

For example, when training a 12-layer encoder with 6 MoE-converted layers over 8 epochs, we proceed as follows:

- Epochs 1–6: progressively convert 1 new layer per epoch.
- Epochs 7–8: full MoE model trained as-is for final optimization.

This approach prevents sudden optimization shocks, allowing each newly introduced expert layer to adapt gradually to the model's dynamics.

Two-Stage Learning: Load Balancing → Specialization. We further split the training process into two functional stages to align with the final goal of pruning underutilized experts:

- **Stage 1 - Balanced Exploration.** During the layer-wise conversion phase (before all target layers are MoE-enabled), we apply a load balancing loss to encourage even expert utilization. This prevents early expert collapse and ensures that each expert receives sufficient gradient signals.

  Unlike simple entropy-based approaches, we compute the load balancing loss via bidirectional KL divergence between the observed expert usage distribution and an ideal uniform prior. For each MoE layer, let $p \in \mathbb{R}^N$ denote the mean routing probability vector over a batch for N experts (computed as softmax(router_logits).mean(dim=0)), and let $u = \frac{1}{N} \cdot \mathbf{1}$ be the uniform target distribution. To ensure numerical stability, we add a small $\epsilon = 10^{-7}$ to both distributions.

  The load balancing loss is defined as:

$$\mathcal{L} = \frac{1}{2} \cdot \mathrm{KL}\left(\log(p+\epsilon) \,\|\, u+\epsilon\right) + \frac{1}{2} \cdot \mathrm{KL}\left(\log(u+\epsilon) \,\|\, p+\epsilon\right)$$

  where $\mathrm{KL}(\cdot \,\|\, \cdot)$ denotes the Kullback-Leibler divergence. For models with multiple MoE layers, we average the load balancing loss across all layers.

  The total training loss in this stage becomes:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \lambda \cdot \mathcal{L}_{\text{balance}}$$

  where λ is a fixed coefficient, set to 0.1 in our implementation.

  Once all target layers are converted to MoE layers (i.e., when current_modified_layers $\geq$ target_modified_layers), we disable $\mathcal{L}_{\text{balance}}$ to allow the gating mechanism to naturally specialize experts in Stage 2.

- **Stage 2 - Expert Specialization.** After all MoE layers are in place, we disable the load balancing objective and allow the gating mechanism to focus solely on task performance. In this phase, experts naturally specialize, and frequently selected experts are reinforced, while unselected ones begin to fade. This directly supports the goal of pruning, where underused experts can later be removed without harming performance. Importantly, the Top-1 gating strategy is preserved throughout to ensure sparsity and modularity.

This two-phase training process bridges the gap between robust early learning and efficient late-stage specialization. It ensures both high-quality optimization and structural sparsity, laying the groundwork for expert pruning in subsequent steps.

3.3 Top-1 Pruning

Building on the Top-1 gating mechanism described in Section 3.1.3, we perform pruning based directly on the observed usage frequencies. Since each input is routed to exactly one expert during training, the gating decisions produce clear selection statistics, enabling a simple usage-driven pruning strategy. Experts that are rarely or never selected are pruned entirely, yielding a compact and efficient encoder.

Usage-based Expert Selection. Each MoE layer routes inputs to one of several expert heads based on a Top-1 gating decision. After training, we analyze validation-time expert usage across all layers by recording how frequently each expert is selected. For each layer, we rank experts by their activation frequency and identify those that contribute meaningfully to inference.

Pruning Policy. For every MoE layer, we retain only the top-m most frequently used experts and discard the rest. The choice of m can be tuned to control the trade-off between model compression and performance retention. Since our architecture uses hard routing (Top-1) and each expert is structurally independent, the unused branches can be safely removed without any additional fine-tuning.

Architecture Simplification. Following pruning, we statically replace each dynamic MoE layer with a simplified deterministic attention layer that only includes the selected experts. This change eliminates the runtime cost of gating and makes inference more predictable and hardware-efficient.

Effectiveness. This approach yields substantial reductions in parameter count and inference latency. Unlike unstructured pruning, our method operates at the architectural level—making the resulting model easy to deploy and analyze. Experimental results show that pruning has minimal impact on downstream task performance while significantly improving throughput and efficiency.

4 Experiment

4.1 GLUE Benchmark Results

We evaluate **Expert Attention** on the **GLUE** benchmark (Warstadt et al., 2019; Wang et al., 2018) using `BERT-base-uncased` (12 layers, 12 heads per layer) as the default backbone unless otherwise specified. All experiments are conducted on a single NVIDIA A8000 GPU (48GB), with each GLUE task evaluated independently for throughput measurement. We fine-tune all models using AdamW with a learning rate of 2×10^{-5}, weight decay 0.01, gradient clipping at max-norm 1.0, and a cosine learning rate schedule with 10% warmup. The batch size is fixed to 64 following standard GLUE fine-tuning practice.

We progressively replace k out of 12 encoder layers with **Expert Attention** modules using Top-1 routing and the two-stage training strategy described in Section 3.1. **After training, underutilized experts are removed through usage-driven *Top-1 pruning*.**

We evaluate seven GLUE tasks: MNLI, QQP, QNLI, SST-2, MRPC, RTE, and CoLA. Besides task accuracy, we report **parameter reduction** (excluding embeddings), **performance retention** relative to the unpruned baseline, and **inference throughput** measured in samples per second (batch size 64).

Result Analysis. Table 1 shows that **Expert Attention** achieves consistent trade-offs between accuracy and efficiency under different pruning ratios. Moderate pruning settings (2/12–6/12) preserve high task accuracy with over 93% average performance retention while substantially improving inference throughput. Even under aggressive pruning (10/12–11/12), the model retains more than 84% of baseline performance while achieving up to 88.5% parameter reduction and 424.1% throughput improvement.

Figure 3 further visualizes the trade-off between accuracy retention and throughput improvement across GLUE tasks. While aggressive pruning introduces moderate performance degradation, throughput consistently increases as more encoder layers adopt the Expert Attention design. In several mildly pruned settings, pruning even yields slight accuracy improvements, suggesting reduced redundancy and improved expert specialization.

These results demonstrate that **Expert Attention** enables effective Transformer compression with limited performance degradation. By combining sparse expert routing with usage-driven pruning, the proposed framework removes redundant computation while preserving competitive downstream performance. Moreover, removing the routing module after pruning eliminates inference-time routing overhead, enabling compact router-free deterministic inference with substantial practical throughput improvements.

Table 1: Performance of different pruning degrees on seven GLUE tasks. All evaluation metrics are reported as accuracy. The first column indicates the pruning ratio (*pruned layers / total layers*). The last three columns summarize the averaged changes across tasks: parameter reduction ratio (excluding embedding layers), performance retention ratio, and throughput improvement ratio compared with the baseline.

Pruning	**GLUE Task Scores (Acc)**							**Param Reduction** (%)	**Performance Retention** (%)	**Throughput** ↑ (%)
	MNLI	**QQP**	**QNLI**	**SST-2**	**MRPC**	**RTE**	**CoLA**			
Baseline	0.8499	0.9098	0.9081	0.9209	0.8480	0.5776	0.8226	-	-	-
2/12	0.8501	0.9086	0.8960	0.9255	0.8162	0.6173	0.8111	16.1	100.1	18.7
4/12	0.8409	0.9071	0.8882	0.9151	0.8039	0.5993	0.7910	32.2	98.6	43.0
6/12	0.8241	0.9094	0.8755	0.9002	0.6838	0.5560	0.7248	48.2	93.1	81.1
8/12	0.7949	0.8955	0.8444	0.8739	0.7034	0.4801	0.6836	64.3	89.2	146.2
10/12	0.7468	0.8755	0.8175	0.8406	0.6838	0.5523	0.6884	80.4	89.6	282.1
11/12	0.6853	0.8572	0.6096	0.8337	0.6838	0.5307	0.6740	88.5	84.4	424.1

Figure 3: Task-wise visualization of pruning effects across the GLUE benchmark under different Expert Attention layer ratios (2/12–11/12). Each subplot corresponds to a GLUE task. Bars (left Y-axis) indicate accuracy changes relative to the baseline, where upward green bars denote performance improvements and downward red bars denote performance degradation. The purple line (right Y-axis) represents throughput improvement (× fold) under router-free inference. The final panel reports the averaged results across all tasks.

4.2 Comparison with Attention Pruning Methods

Building on the setup in Section 4.1, we further evaluate **Expert Attention** against existing attention pruning methods on MNLI using `BERT-base-uncased` under the same experimental configuration.

We compare our method with several representative attention head pruning methods (Li et al., 2021) under the same unpruned-head budget.

The baselines include:

- **Michel et al.** (Michel et al., 2019): gradient-based attention head pruning.
- **Voita et al.** (Voita et al., 2019): auxiliary-loss-guided binary head selection.
- **Pipelined DSP** and **Joint DSP** (Li et al., 2021): iterative and end-to-end dynamic head pruning.
- **STE** (Fan et al., 2019): differentiable head masking via straight-through estimation.

Most existing methods focus solely on attention head pruning through importance estimation or masking strategies. While effective at reducing attention redundancy, they typically preserve the original FFN structure, limiting overall compression efficiency.

In contrast, **Expert Attention** restructures the attention layer into modular expert branches with sparse Top-1 routing. Inactive experts and their associated FFN branches are removed after training, while a lightweight shared *Expander FFN* pre-

Table 2: Accuracy comparison between **Expert Attention** and existing attention pruning methods on MNLI under different unpruned-head budgets. Model size reduction is reported as best baseline vs. ours.

Unpruned Heads	Michel et al.	Pipelined DSP	Voita et al.	STE	Joint DSP	Expert Attention (Ours)	Model Size Reduction (%)
120	0.8046	0.8441	0.8418	0.8459	0.8497	**0.8501**	4.0 / **16.6**
96	0.8424	0.8327	0.8295	0.8393	0.8441	0.8409	8.0 / **32.6**
72	0.8247	0.8295	0.8324	0.8281	0.8348	0.8241	13.0 / **48.6**
48	0.7926	0.7910	0.7608	0.8231	0.8322	0.7949	17.0 / **64.5**
24	0.7082	0.7629	0.3168	0.8220	0.8251	0.7468	19.0 / **80.5**
12	0.4059	0.5629	0.7691	0.7379	0.7974	0.6853	24.0 / **88.5**

serves representational capacity with minimal parameter overhead. This design enables aggressive pruning while maintaining structural consistency and practical inference efficiency.

Result Analysis. Table 2 shows that **Expert Attention** achieves comparable or higher accuracy than existing attention pruning methods while providing substantially greater model compression. With only 12 unpruned heads, our method reduces model size by 88.5%, significantly exceeding the strongest baseline (Joint DSP at 24%) while maintaining competitive performance (68.53 vs. 79.74).

To further evaluate real-world efficiency, we compare against the extreme **Joint DSP configuration with only 12 active heads**. Despite substantially higher compression, **Expert Attention achieves both higher accuracy retention and significantly improved throughput**. Detailed throughput and hardware efficiency results are provided in Appendix Table 4.

These results demonstrate that our MoE-based pruning framework achieves better trade-offs between accuracy, compression, and runtime efficiency than traditional attention pruning strategies, especially in aggressive compression settings.

4.3 Routing Strategy Analysis

As described in Section 3.1.3 and Section 3.3, our method employs *Top-1 routing* during training and applies usage-driven *Top-1 pruning* after training. To evaluate the effectiveness of sparse routing, we conduct an ablation on QQP by comparing Top-k routing strategies with $k \in \{1, 3, 5, 7, 9, 11\}$ under under different pruning ratios (*converted layers / total layers*, 2/12–11/12 encoder layers). Table 3 reports the average accuracy change relative to the unpruned baseline.

The results show that **Top-1 routing and pruning** is the only configuration that avoids performance degradation, even slightly improving average performance relative to the unpruned baseline. In contrast, larger k values introduce consistent accuracy drops due to increased expert overlap and reduced specialization under high compression.

Table 3: Effect of Top-k routing on pruning performance. We report the average difference in accuracy between pruned and unpruned models on QQP, under varying k values and pruning ratios from 2/12 to 11/12.

Top-k	Avg. Performance Difference (QQP)
1	+0.0021 (↑)
3	−0.2490 (↓)
5	−0.2775 (↓)
7	−0.2817 (↓)
9	−0.2769 (↓)
11	−0.3039 (↓)

This trend remains consistent across GLUE tasks, as shown in Figure 3. Top-1 routing maintains stable task performance while enabling substantial throughput improvements, demonstrating its effectiveness under different pruning intensities and task distributions.

In summary, **Top-1 routing and pruning** achieve strong trade-offs between efficiency and task performance across different compression settings. Detailed per-task results are provided in Appendix Table 9.

5 Conclusion

We present **Expert Attention**, a structured pruning framework that reformulates attention heads as independently routed experts. By combining sparse Top-1 routing with usage-driven pruning, the proposed method enables compact router-free deterministic inference after training. Extensive experiments demonstrate strong trade-offs between accuracy, compression, and inference efficiency.

Limitations

Our Expert Attention method has so far been designed and evaluated primarily on encoder-only Transformer models. Extending it to decoder-only or encoder-decoder architectures introduces additional challenges due to the presence of components such as cross-attention and autoregressive mechanisms. These factors complicate direct adaptation and require more sophisticated approaches.

We recognize these challenges and plan to address them in future work by exploring finer-grained gating mechanisms and adaptive pruning strategies tailored to these architectures and specific downstream tasks. This will help broaden the applicability and impact of our method.

References

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *Preprint*, arXiv:1909.11556.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Preprint*, arXiv:2101.03961.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. *Preprint*, arXiv:2004.04037.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. 2025. Moh: Multi-head attention as mixture-of-head attention. *Preprint*, arXiv:2410.11842.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *Preprint*, arXiv:2006.16668.

Jiaoda Li, Ryan Cotterell, and Mrinmaya Sachan. 2021. Differentiable subset pruning of transformer heads. *Transactions of the Association for Computational Linguistics*, 9:1442–1459.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14014–14024.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *Preprint*, arXiv:2005.07683.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *Preprint*, arXiv:1701.06538.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2020. Superglue: A stickier benchmark for general-purpose language understanding systems. *Preprint*, arXiv:1905.00537.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. BERT-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, Online. Association for Computational Linguistics.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39.

A Additional Experimental Results

A.1 Throughput and Hardware Efficiency

Table 4 reports the inference throughput, accuracy, and model compression results of Expert Attention under different pruning configurations on MNLI. All measurements are conducted using the same hardware environment and inference setting with a fixed batch size of 64.

We compare Expert Attention against two representative baselines: the original BERT-base-uncased model without pruning and the aggressively pruned Joint DSP configuration with only 12 active heads. While Joint DSP achieves moderate throughput improvement through attention head pruning, its parameter reduction remains limited because the original FFN structure is largely preserved.

In contrast, Expert Attention consistently achieves substantially higher throughput as the number of converted layers increases. Under the most aggressive setting (layer 11/12), our method reaches 6636.58 samples/s, corresponding to a 332.84% throughput improvement over the original BERT-base model, while simultaneously reducing model size by 88.5%.

Importantly, the observed speedup is not solely due to attention sparsity. By decoupling attention heads into independent expert branches and replacing the original FFN with a lightweight shared Expander FFN, Expert Attention reduces both attention computation and feed-forward overhead. After pruning, the routing module is removed entirely, resulting in static router-free deterministic inference with significantly improved hardware efficiency.

These results demonstrate that Expert Attention provides practical end-to-end acceleration beyond conventional attention head pruning methods, achieving favorable trade-offs between throughput, compression, and downstream task performance.

A.2 Comparison with Distilled Encoders

Distillation-based compact encoders provide a strong alternative route for efficient inference. We therefore compare Expert Attention with DistilBERT to clarify how pruning an existing fine-tuned encoder differs from training a compact distilled model. For Expert Attention, we use the 6/12 pruning setting from Table 1, which provides a balanced operating point with 48.2% parameter reduction and 93.1% average GLUE performance retention. For fair efficiency evaluation, all throughput measurements are conducted on the same hardware platform using an inference batch size of 32.

Result Analysis. Table 5 shows that Expert Attention achieves higher throughput than DistilBERT across all GLUE tasks under the same inference setting. On large-scale tasks such as MNLI and QQP, Expert Attention also obtains competitive or slightly better accuracy. However, DistilBERT remains stronger on smaller or linguistically sensitive tasks such as MRPC, RTE, and CoLA. This suggests that Expert Attention should not be viewed as a universal replacement for distilled encoders. Instead, it provides a complementary compression route: distillation trains a new compact student model, whereas Expert Attention compresses an existing fine-tuned encoder into a static, router-free architecture.

A.3 Generalization to Retrieval Benchmarks

To evaluate whether Expert Attention generalizes beyond GLUE classification, we further apply it to Qwen-Embedding-0.6B and evaluate on six BEIR retrieval benchmarks. Retrieval workloads are particularly challenging for structured pruning because dense retrieval depends on preserving the geometry of the embedding space. We report NDCG@10, parameter reduction, performance retention, and throughput improvement under different pruning ratios.

Result Analysis. Table 6 shows that Expert Attention extends beyond GLUE classification to retrieval-oriented encoder workloads, but the retrieval setting exhibits a sharper accuracy–efficiency trade-off. Under moderate pruning, the model remains usable: pruning 4/28 blocks retains 89.72% of the average NDCG@10 while reducing parameters by 11.08%, and pruning 8/28 blocks retains 82.42% performance with a 22.08% parameter reduction and 19.96% throughput improvement. However, more aggressive pruning leads to substantial degradation, especially on HotpotQA, FIQA, and MSMARCO. This suggests that retrieval models are more sensitive to structural pruning than GLUE classifiers, likely because dense retrieval relies on preserving fine-grained embedding geometry. Therefore, for retrieval workloads, Expert Attention is most suitable under conservative-to-moderate pruning regimes rather than extreme compression.

Table 4: Performance comparison of pruning methods in terms of throughput (batch size 64), accuracy, and model size reduction. The first two rows are baselines (no pruning; extreme head pruning), followed by Expert Attention configurations with progressively more converted encoder layers.

Model (Unpruned Heads)	Throughput (inf/s)	Model Size Reduction (%)	Acc.	Accuracy Retained (%)
bert-base-uncased (144)	1500	–	84.90	–
Joint DSP (12)	≈ 1995 (↑33%)	26	61.79	72.78%
layer 2/12 (120)	1787.13 (↑16.56%)	16.60	**85.01**	**100.12%**
layer 4/12 (96)	2150.64 (↑40.26%)	32.60	84.09	99.05%
layer 6/12 (72)	2603.85 (↑40.26%)	48.60	82.41	97.14%
layer 8/12 (48)	3211.63 (↑109.47%)	64.50	79.49	93.63%
layer 10/12 (24)	4933.59 (↑221.78%)	80.50	74.68	87.97%
layer 11/12 (12)	**6636.58 (↑332.84%)**	**88.50**	68.53	81.11%

Table 5: Throughput and accuracy comparison between Expert Attention and DistilBERT on GLUE tasks. Expert Attention uses the 6/12 pruning setting. All throughput measurements in this table are conducted with inference batch size 32.

Task	Throughput (samples/s)		Speedup	Acc.		ΔAcc.
	Expert Attn.	DistilBERT		Expert Attn.	DistilBERT	
MNLI	2076.19	1746.31	+18.89%	0.8098	0.8174	-0.0076
QQP	2161.81	1815.66	+19.06%	0.9057	0.8938	+0.0119
QNLI	1912.09	1666.88	+14.71%	0.8691	0.8764	-0.0073
SST-2	2153.63	1849.98	+16.41%	0.8911	0.9071	-0.0160
MRPC	1849.22	1624.73	+13.82%	0.7059	0.8235	-0.1176
RTE	1698.56	1509.40	+12.53%	0.5415	0.5884	-0.0469
CoLA	2362.06	993.38	+137.79%	0.7459	0.7910	-0.0451
Avg.	2030.51	1600.33	+33.32%	0.7813	0.8139	-0.0326

B Ablation Studies

B.1 Detailed Top-k Routing Results

Table 9 provides the detailed results across different Top-k routing configurations and pruning settings, corresponding to the averaged statistics reported in Table 3 of Section 4.3. For each Top-k routing configuration, we report the QQP accuracy before and after pruning under settings ranging from 2/12 to 11/12 converted layers.

The detailed results further confirm that Top-1 routing consistently maintains the most stable behavior across different pruning settings. In contrast, larger k values exhibit substantially larger performance degradation after pruning, especially under aggressive compression settings. These observations are consistent with the analysis in Section 4.3 and further support sparse Top-1 routing as the most pruning-friendly configuration in our framework.

B.2 Training Strategy Analysis

We further conduct ablation studies on the proposed training framework, focusing on two key components: the load-balancing objective and the progressive layer-wise conversion strategy. All experiments are performed under the fixed 6/12 pruning setting, which achieves the best overall trade-off between performance retention and compression efficiency in Section 4.1. Since all variants use the same pruning configuration, the parameter reduction ratio and throughput remain nearly identical across settings and are therefore omitted from evaluation. Experimental results are reported in Table 8.

B.2.1 Effect of the Load-balancing Loss

As shown in Table 8, removing the load-balancing loss during Stage 1 (Balanced Exploration) consistently degrades performance on the majority of GLUE benchmarks. This result suggests that balanced expert utilization during the early training stage is important for preventing expert collapse and ensuring sufficient gradient updates for different experts.

In contrast, keeping the load-balancing loss active throughout both training stages before post-training router removal also leads to performance

Table 6: Performance under different pruning ratios on six BEIR retrieval benchmarks using Qwen-Embedding-0.6B as the backbone. All retrieval metrics are reported as NDCG@10. The first column denotes the pruning ratio over all prunable Expert Attention blocks. The last three columns summarize parameter reduction, performance retention, and throughput improvement relative to the unpruned baseline model.

Pruning	BEIR Task Scores (NDCG@10)							Param Reduction (%)	Performance Retention (%)	Throughput ↑ (%)
	NFCorpus	SciFact	HotpotQA	FEVER	FIQA	MSMARCO	Avg.			
Baseline	0.3817	0.7369	0.7834	0.9486	0.3639	0.8952	0.6849	–	–	–
4/28	0.3870	0.6547	0.6344	0.8998	0.3000	0.8121	0.6147	11.08	89.72	4.24
8/28	0.3990	0.5608	0.5065	0.8558	0.2787	0.7251	0.5543	22.08	82.42	19.96
12/28	0.4021	0.5381	0.3537	0.7858	0.2008	0.5603	0.4735	33.16	72.28	39.48
16/28	0.4065	0.5271	0.3567	0.7911	0.2105	0.5533	0.4742	44.16	72.94	49.08

Table 7: QQP grouping results under different pruning degrees. *Pruned Layers* denotes the number of encoder layers converted to Expert Attention and pruned out of 12 BERT-base layers. *Conversion Granularity* denotes the number of layers converted at each progressive conversion step. "Group-x" denotes that x encoder layers are converted at each progressive conversion step..

Pruned Layers	Conversion Granularity	Accuracy
–	Baseline	0.9098
2/12	Group-1	**0.9086**
	Group-2	0.9066
4/12	Group-1	0.9071
	Group-2	0.9082
	Group-4	**0.9085**
6/12	Group-1	**0.9094**
	Group-2	0.9067
	Group-3	0.9050
	Group-6	0.9041
8/12	Group-1	0.8955
	Group-2	**0.9027**
	Group-4	0.8985
	Group-8	0.8988
10/12	Group-1	0.8755
	Group-2	**0.8904**
	Group-5	0.8881
	Group-10	0.8858
11/12	Group-1	0.8572
	Group-2	**0.8724**
	Group-11	0.8679

degradation, although less severe than removing it entirely. We attribute this behavior to the conflict between balanced routing and expert specialization during Stage 2. Once all Expert Attention layers are fully converted, continued load balancing discourages the routing mechanism from concentrating on task-relevant experts, thereby weakening specialization and reducing pruning effectiveness.

These results validate the effectiveness of the proposed two-stage learning strategy: balanced exploration in Stage 1 followed by sparse expert specialization in Stage 2.

B.2.2 Effect of Progressive Conversion Granularity

To further analyze the proposed progressive conversion strategy, we additionally investigate different conversion granularities beyond the default layer-wise setting. Specifically, instead of converting a single encoder layer at each progressive step, we also study group-wise conversion, where multiple layers are converted simultaneously during each stage.

Since all variants under the same pruning configuration contain nearly identical parameter counts and inference complexity, throughput and parameter reduction results are omitted in this section. These metrics have already been analyzed in Table 4 and are primarily determined by the total number of pruned layers rather than the conversion granularity itself.

Table 8 first compares different conversion granularities under the fixed 6/12 pruning setting. We observe that different grouping strategies exhibit distinct task preferences. The proposed layer-wise strategy achieves the strongest performance on large-scale benchmarks such as MNLI, QQP, and QNLI, while several group-wise settings perform competitively on comparatively smaller or higher-variance tasks such as MRPC, RTE, and CoLA. These observations suggest that conversion granularity substantially affects the optimization dynamics after structural transformation.

Motivated by this phenomenon, we further investigate the relationship between grouping strategy and pruning degree on QQP, as shown in Table 7.

Table 8: Ablation results on seven GLUE tasks under the fixed 6/12 pruning setting. The table reports task accuracy under different training strategies and conversion granularities. Parameter reduction and throughput are omitted since all variants use the same 6/12 configuration and therefore have nearly identical model sizes and inference complexity.

Method / Variant	**GLUE Task Scores (Acc.)**						
	MNLI	**QQP**	**QNLI**	**SST-2**	**MRPC**	**RTE**	**CoLA**
Baseline	0.8499	0.9098	0.9081	0.9209	0.8480	0.5776	0.8226
OURS	**0.8241**	**0.9094**	**0.8755**	0.9002	0.6838	0.5560	0.7248
Ablation I: Effect of the Load-balancing Loss							
w/o $\mathcal{L}_{\text{balance}}$	0.8055	0.9053	0.8671	**0.9083**	0.6838	0.5199	0.7267
always-on $\mathcal{L}_{\text{balance}}$	0.8094	0.9050	0.8680	0.9014	0.6838	0.5884	0.6989
Ablation II: Effect of Progressive Conversion Granularity.							
Group-wise conversion (size = 2)	0.8105	0.9067	0.8719	0.9048	0.7353	0.5054	0.6970
Group-wise conversion (size = 3)	0.8154	0.9050	0.8684	0.9060	**0.8186**	**0.6354**	**0.7689**
Group-wise conversion (size = 6)	0.8110	0.9041	0.8627	**0.9083**	0.7206	0.5126	0.7641

For the 11/12 setting, Group-2 denotes a progressive schedule of `2-2-2-2-1`, since 11 layers cannot be evenly divided into groups of size 2. This setting is included to study grouping behavior under extremely aggressive pruning.

Several important trends can be observed. Under relatively mild pruning ratios (e.g., 2/12 and 4/12), the performance differences between grouping strategies remain small, indicating that the model still retains sufficient redundancy to absorb simultaneous structural changes. In these settings, even large-group conversion introduces only limited optimization difficulty.

However, as the pruning ratio increases, the impact of conversion granularity becomes substantially more pronounced. Large-group conversion generally leads to noticeable performance degradation under aggressive pruning settings such as 6/12, 8/12, 10/12, and 11/12. We attribute this behavior to the abrupt distribution shift caused by simultaneously replacing many encoder layers with Expert Attention modules, which makes the optimization process substantially less stable.

Importantly, the proposed layer-wise progressive conversion exhibits the most stable behavior across different pruning settings. In particular, under the representative 6/12 pruning configuration, Group-1 achieves the best performance among all grouping strategies and remains extremely close to the unpruned baseline. This suggests that gradually converting one layer at a time allows the remaining Transformer layers to progressively adapt to the evolving routing dynamics and sparse expert specialization process.

Interestingly, under extremely high pruning ratios (e.g., 8/12–11/12), moderate grouping strategies such as Group-2 occasionally outperform fully layer-wise conversion. One possible explanation is that, in highly compressed settings, excessively fine-grained conversion may slow down global co-adaptation among heavily transformed layers, while moderate grouping provides a better balance between optimization stability and adaptation efficiency.

Overall, these results indicate that conversion granularity plays an important role in stabilizing training after structural pruning. While no single grouping strategy is universally optimal, the proposed layer-wise strategy provides the most robust and consistently competitive behavior across different pruning degrees, particularly under practically useful pruning settings such as 6/12.

Table 9: Detailed accuracy changes under different Top-k routing settings and pruning depths (2–11 out of 12 layers) on QQP. Each entry reports accuracy before and after pruning, along with the difference.

Top-k	**Pruned Layers**	**Pre-Prune Acc.**	**Post-Prune Acc.**	**Diff**
1	2	0.9174	0.9186	+0.0011
	4	0.9094	0.9106	+0.0011
	6	0.9071	0.9094	+0.0023
	8	0.8922	0.8876	−0.0046
	10	0.8429	0.8578	+0.0149
	11	0.8188	0.8165	−0.0023
3	2	0.9209	0.9128	−0.0080
	4	0.9163	0.6628	−0.2534
	6	0.8968	0.5470	−0.3498
	8	0.8853	0.5459	−0.3394
	10	0.8463	0.5757	−0.2706
	11	0.8337	0.5608	−0.2729
5	2	0.9266	0.9002	−0.0264
	4	0.9128	0.6502	−0.2626
	6	0.9002	0.5436	−0.3567
	8	0.8865	0.5298	−0.3567
	10	0.8544	0.5252	−0.3291
	11	0.8429	0.5092	−0.3337
7	2	0.9209	0.8819	−0.0390
	4	0.9094	0.6181	−0.2913
	6	0.9037	0.5413	−0.3624
	8	0.8784	0.5298	−0.3486
	10	0.8578	0.5413	−0.3165
	11	0.8394	0.5069	−0.3326
9	2	0.9255	0.9060	−0.0195
	4	0.9128	0.6239	−0.2890
	6	0.9025	0.5562	−0.3463
	8	0.8979	0.5390	−0.3589
	10	0.8521	0.5206	−0.3314
	11	0.8475	0.5310	−0.3165
11	2	0.9163	0.8486	−0.0677
	4	0.9140	0.5849	−0.3291
	6	0.9014	0.5229	−0.3784
	8	0.8888	0.5264	−0.3624
	10	0.8578	0.5000	−0.3578
	11	0.8326	0.5046	−0.3280