Maximizing LLM Efficiency Through Optimization Strategies

Iman Abbasnejad Tomal Deb Xuefeng Liu
AWS Generative AI Innovation Center

Abstract

As Large Language Models (LLMs) scale in size, their capabilities dramatically improve, but this dimensional expansion simultaneously introduces substantial computational barriers to efficient inference. While various optimization methods exist including model pruning, knowledge distillation, and quantization, their effectiveness and interaction effects remain insufficiently characterized across deployment scenarios. In this work, we perform comprehensive comparisons between inference optimization techniques for LLMs, systematically evaluating their impact on model performance and computational efficiency. Our experiments with Llama3 and Qwen models reveal that knowledge distillation effectively mitigates performance degradation from pruning while caching and hardware acceleration provide complementary benefits. Most significantly, we find that optimally combining these approaches enables smaller models to achieve performance comparable to models 4 × larger while reducing inference latency by up to 100 ×.

CCS Concepts

• **Theory of computation** → *Probabilistic computation.*

Keywords

Inference optimization, Model Pruning, Compression

ACM Reference Format:
Iman Abbasnejad, Tomal Deb, and Xuefeng Liu. 2025. Maximizing LLM Efficiency Through Optimization Strategies. In *Proceedings of The 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '25).* ACM, New York, NY, USA, 8 pages.

1 Introduction

LLMs [22, 33, 34] have demonstrated exceptional capabilities through pre-training [3, 4]. Their superior performance on complex language tasks has led to widespread adoption despite their considerable size compared to smaller NLP based models such as BERT [5] and MicroBERT [11], making inference optimization a critical research area. As the LLM models grow, they exhibit emergent abilities [37] but also present significant computational challenges, requiring substantial resources and incurring high deployment costs. To address these issues, researchers and industries have proposed various optimization methods including model pruning [19, 36, 39], knowledge distillation [29, 30], and quantization [2, 9]. In this work, we perform comprehensive comparisons between different methods of model inference optimization to understand their strengths and limitations in real-world scenarios. We evaluate several approaches including model pruning techniques that remove redundant neural connections, knowledge distillation methods that transfer knowledge from large teacher models to smaller models and hardware accelerators like AWS Inferentia. Our analysis considers not only the traditional metrics like accuracy, perplexity and ROUGE-L metric, but also practical measures including latency across various hardware configurations and quantizations. This multi-dimensional evaluation provides insights into which optimization techniques work best for different deployment scenarios and model sizes. By presenting these comparisons in a systematic way, our work helps researchers and industries make decisions about which inference optimization strategies to adopt based on their specific requirements and constraints.

2 Related Works

Large language models have shown impressive capabilities across numerous tasks, but their parameter-heavy architectures present significant deployment challenges. Prior work has explored various optimization approaches to address latency and resource constraints [21, 31]. Pruning methods [19, 23, 40] aim to remove redundant parameters while preserving model performance. These techniques range from simple magnitude-based approaches [15, 43] that remove small-valued weights to more sophisticated methods. Advanced pruning strategies include first-order importance estimation [18] and hessian-based calculations [19] that more accurately identify critical parameters.Structural pruning [7] has gained particular attention for its hardware compatibility by removing entire structural elements (e.g., attention heads, filters) rather than individual weights, resulting in dense but smaller models that maintain hardware acceleration benefits. Knowledge distillation [30] represents another significant approach for model compression, transferring learned representations from larger teacher models to more compact student architectures [25, 27]. This technique enables smaller models to benefit from the knowledge embedded in larger pre-trained models, often achieving performance closer to the teacher than training the smaller model from scratch. Quantization methods [2, 42] offer complementary benefits by reducing numerical precision while preserving inference capabilities. Recent advances in post-training compression [20] have accelerated optimization by reformulating reconstruction error as linear least squares problems, while layer-wise approaches [8, 10, 14, 28, 38] apply pruning techniques sequentially for efficiency gains. Metric-based methods such as SparseGPT [8] and Sun [28] use Hessian information or weight-activation products to identify important parameters. In parallel, optimization-based methods including L0 regularization [24], CoFi [38], and Compresso [14] learn parameter importance through direct performance feedback during training or fine-tuning. While these techniques have been extensively studied individually, systematic comparisons across multiple dimensions remain limited. Our work addresses this gap by comprehensively evaluating diverse optimization approaches including model pruning, knowledge distillation, and hardware accelerators.

3 Method

In this section, we present our framework for evaluating and comparing various inference optimization techniques for LLMs. Our methodology includes five key approaches: model pruning, knowledge distillation, quantization methods, prompt caching, and hardware acceleration through AWS Inferentia.

3.1 Model Pruning

We use the layer-pruning method from [13], which is based on the observation that representations in transformer networks evolve gradually across layers. In a transformer architecture, each layer follows a residual iteration:

$$x^{(\ell+1)} = x^{(\ell)} + f(x^{(\ell)}, \theta^{(\ell)}) \tag{1}$$

where $x^{(\ell)}$ and $\theta^{(\ell)}$ represent the multi-dimensional input and parameter vectors for layer ℓ, and $f(x, \theta)$ describes the transformation of a multi-head self-attention and MLP layer block. The pruning algorithm identifies layers where representations remain similar across multiple layers, making those intermediate layers potentially redundant. It computes the angular distance between inputs at different layers and removes the layers where this distance is minimized. This is based on the hypothesis that if representations change slowly such that $x^{(\ell)} \approx x^{(\ell-1)} + \epsilon$ with $\epsilon \ll x^{(\ell)}$, then certain layers can be removed with minimal impact on performance. After pruning the redundant layers, fine-tuning can be applied to heal any representation mismatches.

3.2 Knowledge Distillation

This approach [16] is a neural network compression technique where a smaller "student" model learns to mimic the behavior of a larger, more complex "teacher" model. In the process, the student model captures the generalized knowledge embedded within the teacher's learned representations rather than just the hard class labels. This can be done by training the student model to match the teacher's output probability distributions, which are typically "softened" using a temperature parameter, T in the softmax function. The softened distributions (where $T > 1$) reveal more information about the teacher's internal representations than hard labels alone, showing not just which class the teacher predicts but the relative similarities between classes that the teacher has learned. In practice, distillation often uses a weighted combination of two objective functions: matching the teacher's soft probability distributions and predicting the correct hard labels, with the soft targets usually given higher priority. This approach effectively transfers the teacher's generalized knowledge while maintaining task performance, enabling significantly smaller models to achieve comparable accuracy to their larger counterparts

3.3 Model Quantization

Quantization reduces the numerical precision of model weights and activations, decreasing memory requirements and computations while attempting to preserve model behavior. Neural network computations typically use high-precision floating-point formats (FP32) during training, but this level of precision is often unnecessary during inference. By reducing precision, we can achieve efficiency gains with minimal impact on model quality. We evaluate multiple quantization methods across various precision levels including, FP32, BF16, INT8 and INT4.

3.4 Prompt Caching

Prompt caching aims to avoid redundant computation by storing and reusing results from previously processed prompts, particularly effective in applications where similar queries are frequently encountered. The KV (Key-Value) cache optimizes autoregressive generation by storing previously computed attention vectors, $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$, reducing computational complexity from $O(t^2)$ to $O(t)$ for a sequence of length t, though requiring memory that scales as $2 \times L \times h \times d \times t$ bytes for a model with L layers, h attention heads, and hidden dimension d. In this work we use three different prompt caching techniques:

- Static Caching maintains a direct hash table mapping exact prompt strings to their complete KV representations,
- Dynamic Semantic Caching uses dense vector embeddings to compute similarity scores between incoming prompts and cached entries, allowing retrieval of KV values,
- HQQ (Hierarchical Quantized Query) Caching, which combines quantized vector representations with hierarchical indexing to dramatically reduce memory requirements while maintaining rapid retrieval of KV values.

3.5 AWS Inferentia Acceleration

AWS Inferentia [1] is AWS's purpose-built ML inference chip optimized for cost-efficient deep learning deployment. Each Inferentia chip contains four high-performance cores with dedicated memory, while NeuronLink facilitates efficient inter-chip communication. We compile LLMs for this platform to maximize performance advantages.

4 Experiments

In this section we describe our approach to model optimization through knowledge distillation, pruning, quantization, and their combination, as well as our evaluation methodology.

Base Models: For our experiments, we utilize three foundation models as our baselines, Llama3-1B, Llama3-8B [12] and Qwen2.5-32B [32, 41]. These models provide a spectrum of parameter counts and computational requirements, allowing us to analyze trade-offs between model size, inference speed, and performance.

Knowledge Distillation: We implement a knowledge distillation framework where the larger Qwen2.5-32B serves as the teacher model, while the smaller Llama-1B and Llama-8B serve as students.

Model Pruning: Building on the work of [13], we implement model pruning to reduce the computational footprint of our models. Our pruning methodology focuses on identifying and removing redundant components while minimizing impact on model performance.

Combined Approach: To investigate the effects of combining pruning with knowledge distillation, we also perform knowledge

distillation on the pruned variants of Llama3-1B and Llama3-8B. In this setup, we use the full Qwen2.5-32B model as the teacher, and the pruned Llama models as students. This combined approach aims to recover performance lost during pruning while maintaining the inference speed benefits. We hypothesize that knowledge distillation can help compensate for model size reduction caused by pruning, especially for crucial functional components of the model.
Fine-tuning: We follow standard practice for fine-tuning, using a batch size of 16 examples over 3 epochs. The full set of training hyperparameters is available in Appendix A.
Dataset: For training we use Unnatural Instruct dataset [17] which contains $68k$ text instructions. We use the same template as [35] for formatting instructions and inputs. We follow the same methodology used in [17, 35] to create a validation set of $1,000$ examples for model selection. For testing the model performance we use LMentry [6] and test set of Super-Natural Instructions [35].
Evaluation Metrics: We evaluate all models using a comprehensive set of metrics:

- Inference Time: Measured in seconds per sample and tokens per second,
- Perplexity: Assessing the model's ability to predict text by measuring the uncertainty in generating the next token,
- ROUGE-L: Measuring the overlap between model-generated text and reference answers, with ROUGE-L specifically capturing the longest common subsequence.
- LMentry score [6]: Uses greedy decoding and measures the generated outputs using high-accuracy regular expressions.
- EM: Exact match of generated text and their ground truth.
- Accuracy: Fraction of correct predictions.

Our evaluation approach allows us to quantify the practical trade-offs between model size, computational efficiency, and model performance across different optimization techniques.

5 Results

We performed an extensive series of evaluations on the proposed techniques and compare the performance and computational complexity of the methods in this section.

5.1 Base Model Performance

Our analysis begins with the baseline performance of each model without pruning, caching, or knowledge distillation. As shown in Table 1, Llama3-1B achieved an LMentry score of 59.2 with a latency of 2.42s, while Llama3-8B reached 60.1 points at 2.91s, and Qwen2.5-32B attained 65.9 points with the longest latency of 3.74s. On the Super-Natural Instructions benchmark, Table 2, we observe a clearer performance hierarchy, with Qwen2.5-32B achieving 27.8% exact match (EM) and 91.7% accuracy, followed by Llama3-8B (23.5% EM, 84.9% accuracy) and Llama3-1B (18.7% EM, 78.3% accuracy). These results confirm the expected correlation between model size and task performance

5.2 Model Pruning Impact

When applying model pruning, we observe a consistent trade-off between computational efficiency and task performance. At 20% pruning with no other optimizations, Llama3-1B's LMentry score decreased to 51.8 while improving latency to 1.83s (24.4% reduction).

Table 1: LMentry benchmark results with latency improvements. Best latency (yellow) and scores (pink) highlighted. "Impr.(%)" column shows latency reduction vs. baseline (0% pruning, no cache).

Model	Pruning	Cache	Performance		Efficiency		
			Perplexity↓	LMentry↑	Latency↓(s)	Tokens/s↑	Impr.(%)
Llama3-1B	0%	None	15.38	59.2	2.42	1066.7	-
Llama3-1B	0%	Dynamic	15.38	59.2	1.99	1272.0	17.8%
Llama3-1B	0%	HQQ	15.39	59.1	2.24	1233.5	7.4%
Llama3-1B	0%	Static	15.38	59.2	2.32	1180.5	4.1%
Llama3-1B	20%	Dynamic	19.85	51.8	1.83	1445.1	24.4%
Llama3-1B	40%	Dynamic	26.19	43.6	1.53	1633.3	36.8%
Llama3-1B+KD	0%	None	13.92	61.7	2.32	1076.7	-
Llama3-1B+KD	0%	Dynamic	13.92	61.7	1.97	1268.0	15.1%
Llama3-1B+KD	0%	HQQ	13.93	61.6	2.14	1213.5	7.8%
Llama3-1B+KD	0%	Static	13.92	61.7	2.32	1149.5	0.0%
Llama3-1B+KD	20%	Dynamic	17.47	56.3	1.81	1434.1	22.0%
Llama3-1B+KD	40%	Dynamic	22.64	49.4	1.51	1551.3	34.9%
Llama3-8B	0%	None	10.32	60.1	2.91	844.4	-
Llama3-8B	0%	Dynamic	10.32	60.1	2.53	1009.8	13.1%
Llama3-8B	0%	HQQ	10.33	60.0	2.61	984.0	10.3%
Llama3-8B	0%	Static	10.32	60.1	2.69	960.5	7.6%
Llama3-8B	20%	Dynamic	14.78	55.6	2.33	1151.8	19.9%
Llama3-8B	40%	Dynamic	20.92	47.5	1.92	1342.8	34.0%
Llama3-8B+KD	0%	None	9.18	63.5	2.89	879.4	-
Llama3-8B+KD	0%	Dynamic	9.18	63.5	2.42	1029.8	16.3%
Llama3-8B+KD	0%	HQQ	9.19	63.4	2.61	991.0	9.7%
Llama3-8B+KD	0%	Static	9.18	63.5	2.66	990.5	8.0%
Llama3-8B+KD	20%	Dynamic	12.97	59.8	2.25	1162.8	22.1%
Llama3-8B+KD	40%	Dynamic	18.26	52.9	1.94	1310.9	32.9%
Qwen2.5-32B	0%	None	11.54	65.9	3.74	673.1	-
Qwen2.5-32B	0%	Dynamic	11.54	65.9	3.53	795.4	5.6%
Qwen2.5-32B	0%	HQQ	11.55	65.8	3.44	764.4	8.0%
Qwen2.5-32B	0%	Static	11.54	65.9	3.61	749.3	3.5%
Qwen2.5-32B	20%	Dynamic	15.37	56.4	2.79	917.6	25.4%
Qwen2.5-32B	40%	Dynamic	22.11	48.7	2.34	1124.6	37.4%

Similar patterns emerge for Llama3-8B (55.6) and Qwen2.5-32B (56.4), with latency improvements of 19.9% and 25.4%, respectively.

At 40% pruning, more substantial performance degradation occurs, Llama3-1B's LMentry score falls to 43.6, Llama3-8B to 47.5, and Qwen2.5-32B to 48.7. However, the latency improved by 36.8%, 34.0%, and 37.4%, respectively. Notably, larger models appear more resilient to aggressive pruning, maintaining better relative performance at higher pruning rates.

On the Super-Natural Instructions benchmark, Table 2, 20% pruning reduced exact match rates by 2.9 (to 15.8%), 2.8 (to 20.7%), and 3.3 (to 24.5%) percentage for Llama3-1B, Llama3-8B, and Qwen2.5-32B respectively. At 40% pruning, this degradation increased to 6.1 (to 12.6%), 7.1 (to 16.4%), and 8.2 (to 19.6%) percentage. The corresponding decreases in accuracy follow a similar pattern, with reductions of 4.7, 3.6, and 3.7 percentage points respectively. From this analysis we can observe that, the percentage-wise reduction in exact match rate is more severe than the reduction in accuracy, suggesting that pruning primarily affects the model's precision in generating exact outputs rather than its overall understanding of instructions.

5.3 Caching Strategies

Without changing the model architecture, caching strategies offers pure efficiency gains without much performance losses. As shown in Table 1, dynamic caching provides substantial latency improvements across all models: 17.8% for Llama3-1B (2.42s to 1.99s), 13.0% for Llama3-8B (2.91s to 2.53s), and 5.6% for Qwen2.5-32B (3.74s to 3.53s). Importantly, these latency reductions come with not much decline in performance metrics, preserving identical LMentry scores, exact match rates, and accuracy percentages.

Table 2: Super-Natural Instructions results with EM rate changes. Best accuracy highlighted (pink). EM Δ(%) shows change relative to baseline (0% pruning, no KD).

Model	Pruning	Cache	Performance			
			R-L↑	EM↑(%)	Acc↑(%)	EM Δ (%)
Llama-1B	0%	None	0.31	18.7	78.3	-
Llama-1B	20%	None	0.30	15.8	73.6	-15.5%
Llama-1B	40%	None	0.25	12.6	68.2	-32.6%
Llama-1B+KD	0%	None	0.34	21.3	83.7	+13.9%
Llama-1B+KD	20%	None	0.33	19.1	80.6	+2.1%
Llama-1B+KD	40%	None	0.29	15.8	74.5	-15.5%
Llama-8B	0%	None	0.36	23.5	84.9	-
Llama-8B	20%	None	0.33	20.7	81.3	-11.9%
Llama-8B	40%	None	0.26	16.4	75.8	-30.2%
Llama-8B+KD	0%	None	0.38	26.7	89.5	+13.6%
Llama-8B+KD	20%	None	0.36	24.3	87.1	+3.4%
Llama-8B+KD	40%	None	0.30	20.2	82.4	-14.0%
Qwen2.5-32B	0%	None	0.39	27.8	91.7	-
Qwen2.5-32B	20%	None	0.35	23.5	88.0	-15.5%
Qwen2.5-32B	40%	None	0.28	19.6	82.1	-29.5%

HQQ caching offers varied improvements: 7.4% for Llama3-1B, 10.3% for Llama3-8B, and 8.0% for Qwen2.5-32B. This strategy incurs minimal performance penalties ($0.1to0.2$ on LMentry), making it a viable alternative when memory efficiency is prioritized. Static caching provides the most modest efficiency gains of $4.1to7.6$% across models, with Qwen2.5-32B showing only 3.5% improvement, but maintains performance parity with the no-cache baseline.

When analyzing caching strategies across pruned models, we observe that the relative efficiency gains remain consistent regardless of pruning level. For example, caching strategies consistently provide additional latency gains regardless of pruning level, with dynamic caching reducing latency by $15-18$% for Llama3 models and $5-7$% for Qwen2.5-32B across different pruning rates.

5.4 Knowledge Distillation

Knowledge distillation can be used to recover performance lost through pruning while maintaining efficiency gains. Unpruned Llama3-1B+KD achieved an LMentry score of 61.7 (from 59.2), and close to the much larger Llama3-8B. Similarly, Llama3-8B+KD reached 63.5 from 60.1. On the Super-Natural Instructions benchmark, knowledge distillation yielded significant gains in EM: from 18.7% to 21.3% for Llama3-1B and from 23.5% to 26.7% for Llama3-8B. This brings the Llama3-8B+KD model to within 1.1% of Qwen2.5-32B's performance despite being four times smaller.

The most significant benefit of knowledge distillation appears when combined with pruning. At 20% pruning, Llama3-1B+KD achieved an LMentry score of 56.3, recovering 4.5 points of the 7.4-point loss from pruning. Similarly, Llama3-8B+KD reached 59.8, recovering 4.2 points and nearly matching the unpruned base Llama3-8B. Similar pattern is observed on the Super-Natural Instructions benchmark, where knowledge distillation at 20% pruning improved EM from 15.8% to 19.1% for Llama3-1B and from 20.7% to 24.3% for Llama3-8B. Notably, Llama-8B+KD with 20% pruning achieved 87.1% accuracy, higher than Llama3-8B with no pruning (84.9%) and approaching Qwen2.5-32B with 20% pruning (88.0%). Even at 40% pruning, distilled models maintained substantially better performance: Llama3-1B+KD scored 49.4 on LMentry (vs. 43.6 without KD) and Llama-8B+KD scored 52.9 (vs. 47.5 without KD), demonstrating that knowledge distillation can mitigate performance degradation even at higher pruning ratios.

Table 3: Inference times across quantization methods. INT4 Impr. shows latency reduction vs. baseline; Inf. Impr. shows Inferentia speedup.

Model	Pruning	None	BF16	INT8	INT4	Inferentia	INT4 Impr.	Inf. Impr.
Llama-1B		2.42	0.97	1.01	0.94	0.012	61.2%	201.7×
Llama-1B	20%	1.93	1.02	1.06	0.96	0.007	50.3%	41.7%
Llama-8B		2.91	1.32	2.03	0.91	0.043	68.7%	67.7×
Llama-8B	20%	2.05	1.97	2.01	1.98	0.037	3.4%	14.0%
Qwen2.5-32B		3.74	2.66	2.78	2.30	0.872	38.5%	4.3×
Qwen2.5-32B	20%	3.65	3.46	3.72	3.66	0.641	-0.3%	26.5%

5.5 Hardware Acceleration and Quantization

Table 3 presents inference time across different quantization methods and hardware platforms. All quantization experiments were conducted on NVIDIA A100 GPUs with 2×40GB GPUs. Quantization alone provides significant acceleration, with INT4 quantization reducing latency by 61.2% for Llama-1B (from 2.42s to 0.94s), 68.7% for Llama3-8B (from 2.91s to 0.91s), and 38.5% for Qwen2.5-32B (from 3.74s to 2.30s) compared to unquantized baselines. However, the most dramatic improvements come from hardware acceleration with AWS Inf2 (with 1 chip, 32 cores, 32 NeuronCores, 128GB memory), which delivers 201.7 × for Llama3-1B (from 2.42s to 0.012s), 35.8× for Llama3-8B (from 1.54s to 0.043s), and 3.6 × for Qwen2.5-32B (from 3.12s to 0.872s). When combining Inferentia with model pruning, we observe additional performance gains, with pruned models achieving latency reductions of 41.7% (Llama3-1B), 14.0% (Llama3-8B), and 26.5% (Qwen2.5-32B) compared to their unpruned counterparts on the same hardware.

5.6 Analysis

The full potential of the optimization techniques emerges when combining all three approaches. Llama3-8B+KD with 20% pruning and dynamic caching demonstrates the most favorable balance: it achieves 96.4% of original Qwen2.5-32B's LMentry performance (59.8 vs. 65.9) while reducing latency by 39.8% (2.25s vs. 3.74s). Furthermore, Llama3-8B+KD with 40% pruning and dynamic caching maintains strong performance (52.9 LMentry score, 82.4% accuracy) while delivering a 48.1% latency reduction compared to the Qwen2.5-32B baseline (1.94s vs. 3.74s).

When the models are running on Inferentia hardware, the Llama3-8B+KD model with 20% pruning on Inferentia achieves inference times of 0.037s, approximately 101.1 × faster than the baseline Qwen2.5-32B configuration while maintaining 90.7% of its LMentry performance and 87.4% of its exact match rate. This shows, choosing a right caching and optimization technique, smaller models can achieve performance comparable to models 4 × their size while offering dramatically better efficiency profiles.

6 Conclusion

In this work, we extensively evaluated the impact of various inference optimization techniques on LLM performance and efficiency. Our evaluation reveals that knowledge distillation, model pruning, and caching strategies provide complementary benefits when properly combined. Distillation effectively mitigates performance degradation from pruning, with student models maintaining over 90% of the teacher model's performance despite having 4× fewer parameters. By strategically combining these approaches, we demonstrate that smaller models can match the capabilities of much larger counterparts while dramatically improving inference efficiency.

References

[1] Amazon Web Services. 2023. AWS Inferentia: High-performance Machine Learning Inference. https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/arch/neuron-hardware/inferentia.html. Accessed: 2023-10-15.

[2] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2020. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701* (2020).

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*. 4171–4186.

[6] Avia Efrat, Or Honovich, and Omer Levy. 2022. Lmentry: A language model benchmark of elementary language tasks. *arXiv preprint arXiv:2211.02069* (2022).

[7] Determine Filters'Importance. 2016. Pruning Filters for Efficient ConvNets. (2016).

[8] Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*. PMLR, 10323–10337.

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323* (2022).

[10] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan-Adrian Alistarh. 2023. OPTQ: Accurate post-training quantization for generative pre-trained transformers. In *11th International Conference on Learning Representations*.

[11] Luke Gessler and Amir Zeldes. 2022. MicroBERT: Effective Training of Low-resource Monolingual BERTs through Parameter Reduction and Multitask Learning. In *Proceedings of the The 2nd Workshop on Multi-lingual Representation Learning (MRL)*. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates (Hybrid), 86–99. https://aclanthology.org/2022.mrl-1.9

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[13] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. [n. d.]. The unreasonable ineffectiveness of the deeper layers, 2024. *URL https://arxiv. org/abs/2403.17887* ([n. d.]).

[14] Song Guo, Jiahang Xu, Li Lyna Zhang, and Mao Yang. 2023. Compresso: Structured pruning with collaborative prompting learns compact large language models. *arXiv preprint arXiv:2310.05015* (2023).

[15] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[17] Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv preprint arXiv:2212.09689* (2022).

[18] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems* 33 (2020), 9782–9793.

[19] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. 2022. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259* (2022).

[20] Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. *Advances in Neural Information Processing Systems* 35 (2022), 24101–24116.

[21] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[22] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2023. Bloom: A 176b-parameter open-access multilingual language model. (2023).

[23] Zejian Liu, Fanrong Li, Gang Li, and Jian Cheng. 2021. EBERT: Efficient BERT inference with dynamic structured pruning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. 4814–4823.

[24] Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning sparse neural networks through L_0 regularization. *arXiv preprint arXiv:1712.01312* (2017).

[25] Xinyin Ma, Xinchao Wang, Gongfan Fang, Yongliang Shen, and Weiming Lu. 2022. Prompting to distill: Boosting data-free knowledge distillation via reinforced prompt. *arXiv preprint arXiv:2205.07523* (2022).

[26] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[27] Ahmad Rashid, Vasileios Lioutas, Abbas Ghaddar, and Mehdi Rezagholizadeh. 2020. Towards zero-shot knowledge distillation for natural language processing. *arXiv preprint arXiv:2012.15495* (2020).

[28] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695* (2023).

[29] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355* (2019).

[30] Siqi Sun, Zhe Gan, Yu Cheng, Yuwei Fang, Shuohang Wang, and Jingjing Liu. 2020. Contrastive distillation on intermediate representations for language model compression. *arXiv preprint arXiv:2009.14167* (2020).

[31] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984* (2020).

[32] Qwen Team. 2024. Qwen2.5: A Party of Foundation Models. https://qwenlm.github.io/blog/qwen2.5/

[33] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. 2022. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239* (2022).

[34] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[35] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705* (2022).

[36] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732* (2019).

[37] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).

[38] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694* (2023).

[39] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408* (2022).

[40] Dongkuan Xu, Ian EH Yen, Jinxi Zhao, and Zhibin Xiao. 2021. Rethinking Network Pruning–under the Pre-train and Fine-tune Paradigm. *arXiv preprint arXiv:2104.08682* (2021).

[41] 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, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024. Qwen2 Technical Report. *arXiv preprint arXiv:2407.10671* (2024).

[42] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. *Advances in Neural Information Processing Systems* 35 (2022), 27168–27183.

[43] Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. 2021. Prune once for all: Sparse pre-trained language models. *arXiv preprint arXiv:2111.05754* (2021).

A Detailed Explanation of Experiment

In this section we provide more details of our experiments and findings. For the fine-tuning experiments, we applied a consistent set of hyperparameters across all conditions. Models were trained for a maximum of 10, 000 steps. We selected the final model based on Rouge-L performance on our validation set, evaluating checkpoints at 1000-step intervals. Our training configuration used a batch size of 16 and employed a learning rate schedule with a maximum value of 10^{-5}, incorporating a warm-up phase during the initial 10% of training steps. Weight decay was set at 0.01. We truncated input sequences at 1, 024 tokens and limited output sequences to 128 tokens. All training runs utilized DeepSpeed's ZeRO-3 optimization [26] for efficient distributed training. For fine-tuning we employed 8 NVIDIA V100 40GB GPUs. For evaluation metrics, including Rouge-L and exact match scores, we used the implementation described in [35].

Figure 1 shows the fine-tuning and corresponding validation losses of the proposed base models on the training dataset. Figures 2,3 and4 illustrate the fine-tuning and validation losses for the proposed based models under different layer pruning conditions (0%, 20%, and 40%). All models exhibit widening gaps between training and validation losses. This pattern directly correlates with the EM rate declines described in Table 2, where Llama3-1B shows a 15.5% EM reduction at 20% pruning and 32.6% at 40% pruning. In addition, when moving from 0% to 20% pruning, Llama3-1B's perplexity increased, which is reflected in the higher final loss values.

On the other hand, the visualized losses demonstrate model scale-dependent patterns. Larger models (particularly Qwen2.5-32B) demonstrate smoother convergence trajectories even under pruning conditions, while smaller models (Llama3-1B) show more erratic behavior, especially at 40% pruning. As pruning increases, the gap between training and validation loss widens more dramatically for smaller models than larger ones. This widening gap explains why Qwen2.5-32B maintains 82.1% accuracy at 40% pruning while Llama3-1B drops to 68.2%. Furthermore, the loss curves indicate that all models initially converge at similar rates regardless of pruning level, but diverge significantly in later iterations, suggesting that pruned models have less capacity to capture fine-grained patterns during extended training.

Figure 5 presents layer similarity patterns through heat maps across our pruned models. Each square is colored to represent the row-normalized angular distance between layer ℓ and $\ell + n$ for all possible values of ℓ, with block sizes n extending to substantial fractions of total model depth. Our analysis reveals consistent patterns that directly explain the performance impacts observed in our pruning experiments.

The smallest angular distances (yellower regions) predominantly appear in deeper blocks, indicating that deeper layers typically exhibit higher similarity and thus present better pruning candidates. This pattern aligns with our empirical results in Tables 1 and 2, where larger models like Llama3-8B and Qwen2.5-32B maintain better relative performance at higher pruning rates $(20\% - 40\%)$ compared to the smaller Llama3-1B. The resilience of these larger models to pruning can be attributed to greater redundancy in their deeper layers, as visualized in our similarity maps.

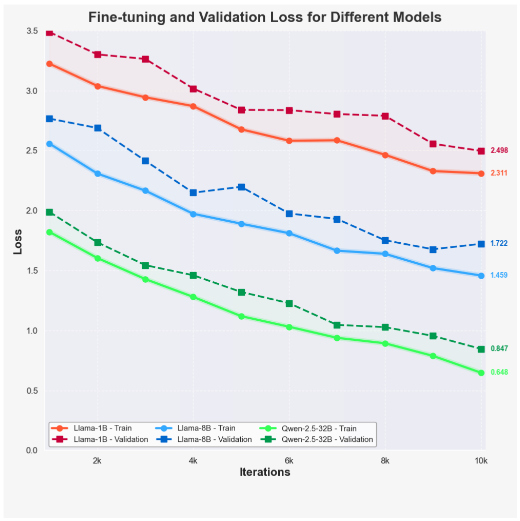

Figure 1: Base model fine-tuning on Unnatural Instruct dataset [17].

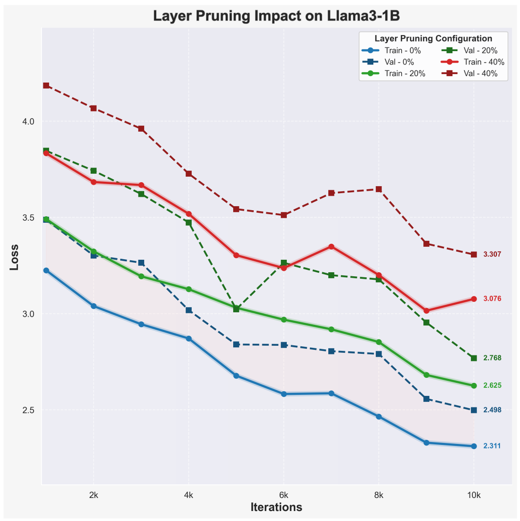

Figure 2: Pruned Llama3-1B model fine-tuning on Unnatural Instruct dataset [17].

The blocks containing the final layer (visible along the outer diagonal) consistently display maximal or near-maximal distance values (red regions), suggesting that the output layer captures unique representations critical to model performance. This empirical finding explains why our pruning strategy deliberately preserves the final layer, allowing us to achieve significant latency improvements $(24.4\% - 37.4\%)$ while minimizing performance degradation.

Interestingly, the Qwen2.5-32B model exhibits distinctive similarity patterns compared to the Llama family. We observe several shallow blocks in Qwen (yellow regions), particularly in the lower-left quadrant of its heatmap. This architecture-specific characteristic explains Qwen's different robustness observed in our experiments, where it experiences a more substantial performance drop at 20%

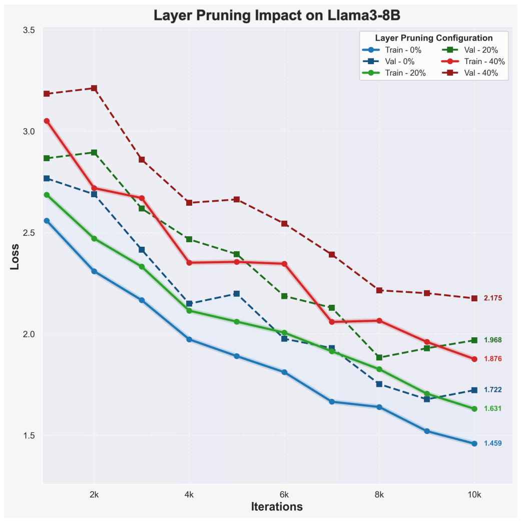

Figure 3: Pruned Llama3-8B model fine-tuning on Unnatural Instruct dataset [17].

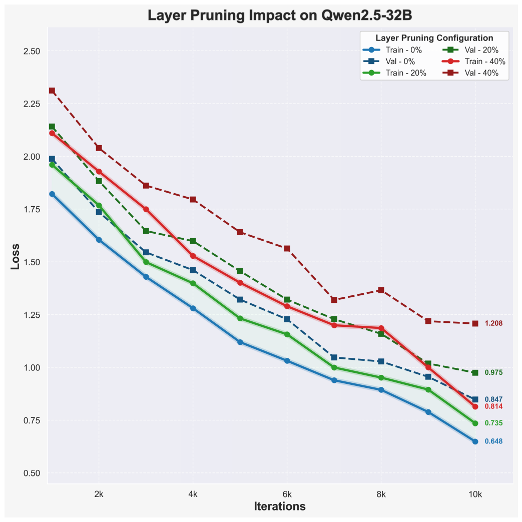

Figure 4: Pruned Qwen2.5-32B model fine-tuning on Unnatural Instruct dataset [17].

pruning (LMentry score decreasing from 65.9 to 56.4, and exact match rate dropping from 27.8% to 24.5%). Despite starting with the highest performance, Qwen's unique layer similarity distribution makes it more sensitive to pruning in certain regions, resulting in less predictable performance degradation compared to the more uniformly structured Llama models.

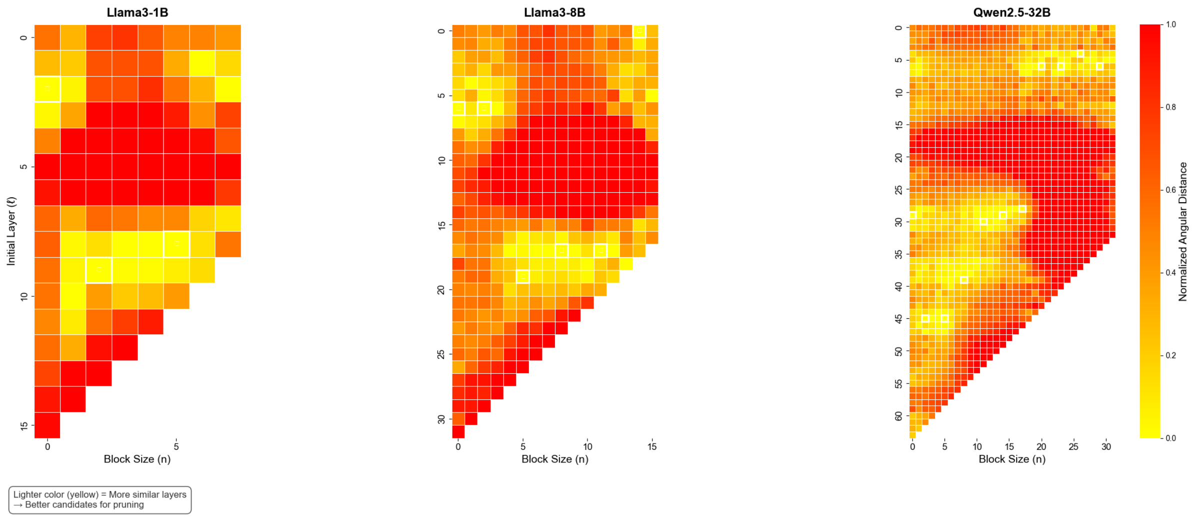

Figure 5: Layer similarity distance between layers. x-axis shows the block size and y-axis shows the initial layers. The distances are rescaled to $[0, 1]$ range. The optimal pruning candidate layers $\ell^*(n)$ appear as deepest yellow in each row.