TELL-TALE: Task Efficient LLMs with Task Aware Layer Elimination

Anonymous ACL submission

Abstract

Large Language Models (LLMs) are typically deployed using a fixed architecture, despite growing evidence that not all layers contribute equally to every downstream task. In this work, we introduce TALE (Task-Aware Layer Elimination), an inference-time method that improves task performance by selectively removing layers that are irrelevant or detrimental for a given task. TALE optimizes task-specific validation performance, yielding a task-adapted architecture without retraining or modifying model weights. Across 9 tasks and 5 model families, under both zero-shot and few-shot settings, we show that TALE consistently matches or surpasses baseline performance while simultaneously reducing computational cost, outperforming general and layer-wise pruning approaches such as SLEB. Beyond inference-time gains, TALE synergizes with fine-tuning and few-shot learning, where task-adapted architectures lead to additional performance improvements. Computing TALE for a new task requires modest resources (1–2 GPU hours on an A100), making it a practical and deployable solution for task-specialized LLM inference.

1 Introduction

The substantial computational costs of Large Language Models (LLMs) can prevent resource-constrained organizations and those with high-throughout applications from leveraging more capable models. This has led to a search for methods that boost task-specific performance and reduce computation costs. The use of multi-agent systems, with LLMs specialized for a particular role, has intensified the need for such methods. Yet finding such methods has proved elusive. Fine-tuning can increase task performance but does not reduce inference costs and requires significant training overhead and data. General pruning reduces computation costs but typically demands significant retraining and often results in substantial performance degradation on downstream tasks.

TALE, Task Aware Layer Elimination addresses this need. TALE increases task performance while modestly reducing computational overhead. TALE evaluates all possible single-layer removals at each iteration, selecting the layer whose elimination results in the highest validation accuracy. This process repeats until performance falls below a user predefined threshold.

TALE is a simple, hardware agnostic, greedy algorithm requiring **no retraining** that runs at inference time. (Peer et al., 2022)'s similar greedy approach, on the other hand, requires retraining. Unlike all pruning methods of which we are aware, TALE optimizes for task-specific accuracy at each pruning step. We believe this is why TALE significantly **improves results over the original model and interacts synergistically with fine-tuning**.

Layer analysis on earlier models observed that not all layers contribute equally to a given task(De Vries et al., 2020; Dalvi et al., 2020; Sajjad et al., 2023). Our study of state of the art, middle-sized transformers sharpens these observations: **the layers most or least useful for a given LLM are highly dependent on the target task**. These findings validate the attractiveness of TALE's simple, task-specific, accuracy-driven design.

We showcase TALE 's gains on five modern LLMs, LLaMA 3.1 8B, Qwen 2.5 7B, Qwen 2.5 0.5B, Mistral 7B and Lucie 7B, with 9 diverse benchmark datasets (Sections 4 and 5). We also compare TALE to previous training-free pruning methods on two older models (Llama 2 7B and 13B) and show that TALE achieves substantially higher accuracy with efficiency gains. With these benchmarks and model families, we show that removing task-misaligned layers can reduce inference cost and, in many cases, improve accuracy. Importantly, these improvements arise without retraining, architectural modification, or changes to

the underlying model weights, and persist under alternative data splits, random seeds, and evaluation protocols. TALE complements fine-tuning and applies post hoc to existing checkpoints, making it a practical tool for deploying task-specialized LLMs.

2 Related work

While there is considerable work on model speedup through various pruning methods (pruning sparsity, acceleration, model compression), very few methods concentrate on boosting accuracy and almost no work has used compression methods to **improve** performance, which is what TALE does.

Structured pruning, which eliminates entire components such as neurons, attention heads, or layers (He et al., 2017; Voita et al., 2019; Lagunas et al., 2021) is relevant to TALE . (Frantar and Alistarh, 2023a; Zhang et al.) prune contiguous blocks with minimal performance loss. Similar to earlier methods that use representation similarity (Dalvi et al., 2020), **SLEB** (Song et al., 2024) removes entire layers based on the cosine similarity of their representations, which is the closest layer-wise baseline to TALE . SLEB's layer selection is validated using a general metric (perplexity) on a linguistic dataset. **SliceGPT** (Ashkboos et al., 2024) prunes layer dimensions via Principal Component Analysis. However, representational similarity, whether measured by cosine similarity, Center Kernel Alignment (Dalvi et al., 2020), or variance under PCA (Sajjad et al., 2023) is not a good measure for assessing a layer's importance for a task. In Section 5.1 we show that representational similarity and validation with perplexity are sub-optimal for maximizing performance on specific downstream tasks.

SparseGPT (Frantar and Alistarh, 2023b) and **Wanda** (Sun et al., 2023) perform weight-level unstructured pruning based on local reconstruction criteria or magnitude-activation products, achieving high sparsity but often degrading task-specific reasoning and linguistic abilities. Methods like OWL(Yin et al., 2024) and FLAP (An et al., 2024) use non uniform sparsity pruning to achieve important efficency gains, while (Tang et al., 2025) uses an evolutionary algorithm and a fitness criterion on a selection of data to find models with a given sparsity. None of these approaches provide real improvements on accuracy for standard benchmarks, and for the most part they show significantly decreased performance from the baseline.

(Peer et al., 2022) use a preset accuracy goal and a greedy algorithm like ours and show that their greedy-layer pruning (GLP) outperforms Top-layer pruning (Dalvi et al., 2020) in almost all cases. Although superficially similar, GLP is very different from TALE. First, GLP **retrains the pruned model on a downstream task** T and compare the retrained model's results on T to the those of the baseline model. TALE, on the other hand, involves **no retraining**. Even with retraining GLP's reported accuracies for GLP models are far below our best models. Second, GLP restricts its algorithm to remove an *a priori* number of layers irrespective of optimization. TALE selects the number of layers that optimizes performance without setting a number *a priori* of layers to remove.

TALE differs from other training free methods in granularity and optimization objective: **TALE is a structured, layer-level method that is explicitly optimized for task-specific accuracy**, whereas the aforementioned training-free baselines are designed for compression using predefined limits and linguistic or reconstruction criteria. Section 5 compares various approaches with TALE , showing that TALE consistently and substantially *improves* accuracy over the unpruned baseline, a benefit not reliably achieved by other methods.

3 Basics and Intuitions

A transformer maps a sequence of input vectors $(x_1, \cdots, x_n)$ to a corresponding sequence of output vectors through a stack of L layers. Each layer ℓ transforms the hidden representations $X^{(\ell)} = (x_1^{(\ell)}, \ldots, x_n^{(\ell)})$ into $X^{(\ell+1)}$ through attention and feedforward blocks, connected by residual pathways. Removing layer ℓ from this pipeline simply redirects the flow such that $X^{(\ell-1)} \to X^{(\ell+1)}$, a property that makes the architecture naturally amenable to layer-wise pruning.

Our intuition for TALE came from examining the behavior of partial forward passes. Let $h^{(k)}$ denote the hidden representation after k layers. Instead of always decoding from the final representation $h^{(\mathrm{L})}$, we projected intermediate representations $h^{(k)}$ for $k < \mathrm{L}$ directly into the vocabulary space using the output projection W_{out}, i.e.,

$$\hat{y}^{(k)} = \mathrm{softmax}(W_{\text{out}} h^{(k)}).$$

When comparing the performance of $\hat{y}^{(k)}$ across different values of k, we surprisingly observed that for many tasks, intermediate layers ($k < \mathrm{L}$)

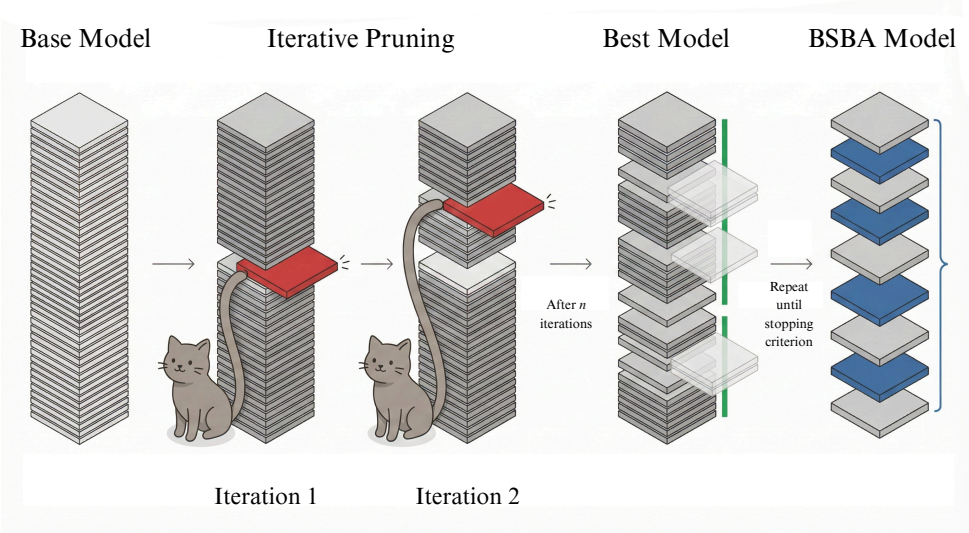


Figure 1: How TALE works to produce BEST and Best Speedup above Baseline (BSBA) models.

achieved higher accuracy than the final layer L (Figure 7). Thus, additional layers did not always improve task-specific performance: some layers contribute marginally, while others introduce representational noise. Not all layers in an LLM are equally useful, and selectively removing redundant layers can preserve—or even improve—downstream accuracy. TALE (Task-Aware Layer Elimination) formalizes this intuition into a principled, iterative optimization strategy.

Algorithm 1 TALE : Iterative Layer Pruning

Require: Pre-trained model $\mathcal{M}$ with L layers; validation set $\mathcal{D}_{val}$; performance threshold ϵ
Ensure: Compressed model $\mathcal{M}^*$

```
Initialize M* ← M
repeat
    for each layer ℓ ∈ {1, ..., L} of M* do
        Construct candidate model M_{-ℓ} by removing layer ℓ
        Compute validation accuracy A_ℓ = Acc(M_{-ℓ}, D_val)
    end for
    Select ℓ* = arg max_ℓ A_ℓ
    if A_{ℓ*} ≥ Acc(M*, D_val) − ε then
        Update M* ← M_{-ℓ*}
    else
        break
    end if
until All Accuracies below threshold
return M*
```

3.1 TALE

TALE is a greedy, iterative layer pruning algorithm for pre-trained open-weights LLM compression that systematically removes layers while preserving or even improving model performance (Algorithm 10). Starting with a full pre-trained model, TALE evaluates all possible single-layer removals at each iteration, computing the validation accuracy for each candidate pruned architecture. The layer whose removal results in the highest accuracy is permanently eliminated from the model, and this compressed architecture becomes the baseline for the next iteration. This process continues iteratively until the performance improvement falls below a predefined threshold. For this paper we take baseline task accuracy - 8% to be the threshold, which allows the accuracy during our search to go slightly below baseline to capture eventual, substantial increases. We have found no cases where the trajectory later goes above the baseline. Once the threshold is reached, the algorithm terminates and returns the most compressed model with performance above the threshold.

Task performance optimization trajectories Figure 2 visualizes three iterative model optimization processes for LLaMA 3.1 8B with TALE . Each curve tracks accuracy as layers are progressively removed. The graphs reveal a general pattern across almost all our tasks: the first iteration of TALE typically provides a large boost in accuracy, followed by slight increases or decreases; they then follow a monotonic decreasing path to accuracies below the baseline and eventually to 0. A full set of curves is

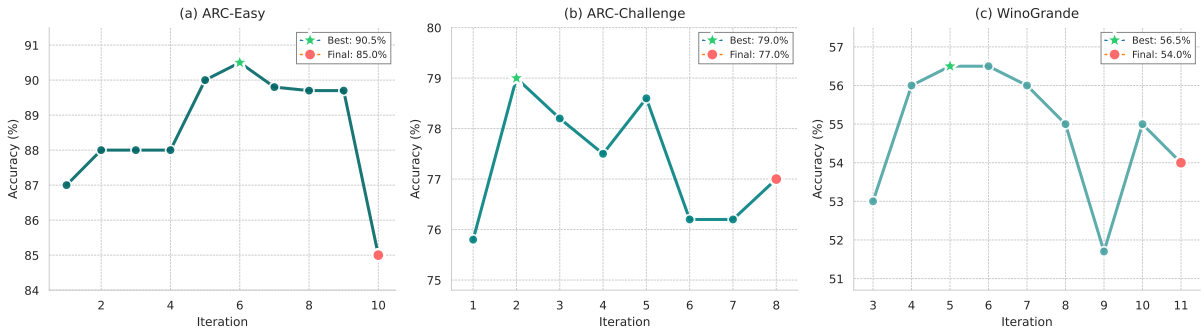


Figure 2: Accuracy progression of TALE across 3 benchmark datasets for LLaMA 3.1 8B. Each curve represents the accuracy at successive iterations. The ⋆ denotes the best-performing layer drop configuration, while the ● highlights the Best Speed up with at least Baseline Accuracy (BSBA) configuration. Plots for all tasks are in Appendix E.

in Figure 11. The curves in themselves are worthy of future study.

Three consistent patterns emerge from these trajectories: (i) TALE identifies compressed **BEST models** that *outperform* the original across diverse tasks, with ⋆ markers lying strictly above baseline. **Best Speed up with at least Baseline Accuracy (BSBA)** are the models with best speed up above TALE's stopping point (with ●). (ii) Accuracy improvements persist across multiple pruning steps before diminishing returns, showing that substantial redundancy exists even in carefully tuned pretrained models. (iii) Pruning dynamics are task-specific: datasets such as ARC-Easy and MMLU tolerate deeper pruning while continuing to improve, whereas reasoning-heavy tasks like GSM8K-Hard converge earlier, reflecting heterogeneous layer importance across domains.

4 Benchmarks and Datasets

We evaluate TALE across a diverse suite of nine benchmarks spanning reasoning, language understanding, and commonsense knowledge. For mathematical reasoning, we use **GSM8K-Hard**, a curated subset of GSM8K (Cobbe et al., 2021) with more than five premises per question to increase difficulty, and **MATH500** (Hendrycks et al., 2021b). (for evaluation details see Appendix A). For language understanding commonsense reasoning and multi-task generalization, we use **MMLU** (Hendrycks et al., 2021a) and **BoolQ** (Clark et al., 2019); **Winogrande** (Sakaguchi et al., 2021), **CommonsenseQA** (Talmor et al., 2019), and**BIG-Bench** (Srivastava et al., 2023), **ARC-Easy** and **ARC-Challenge** (Clark et al., 2018).

In our study TALE requires only modestly-sized sets for task-specific optimization, ranging from 500 to 1500 examples. As seen in Table 3 (Appendix C), once the validation set size exceeds 500 examples, the set of layers dropped stabilizes across all tasks. For each benchmark, we partition the available data into three disjoint subsets: (i) a training pool, (ii) a held-out *optimization split* drawn from the training pool, and (iii) a final *evaluation split* (the official test set when available, or a held-out validation set otherwise).

TALE uses *only* the optimization split to select which layers to remove for a given task. The evaluation split is never used during layer selection. All results reported throughout the paper correspond to performance on this unseen evaluation split.

When official test sets are unavailable, we follow prior work and report results on held-out validation splits that are disjoint from the data used for layer optimization. To see results with two different training subsets see Tables 2 and 17.

5 Results

We emphasize that all TALE pruning decisions are made using held-out optimization splits drawn from the training data only. Test (or final validation) sets are never used for layer selection and are accessed exactly once for evaluation. This protocol ensures that reported gains reflect genuine generalization rather than selection bias.

5.1 Comparisons

We first compare TALE 's best models to state of the art alternatives on two older models. We then analyze our method's robustness on five more modern models.

Most of the state of the art methods for training-free pruning used Llama2-7b and Llama2-13b with LM Eval evaluation (accuracy). Figure 3 shows

that state of the art methods never surpassed the baseline accuracies on the tasks we measured. On the other hand TALE produced best models that consistently surpassed baseline accuracies on the task. Prior work on structured layer pruning, most notably SLEB, has already evaluated naive depth-based baselines such as removing the top-k or bottom-k transformer layers and demonstrated that these strategies are consistently inferior to task- or redundancy-aware pruning. Since our focus is on task-aware layer selection rather than re-criticizing uniform depth trimming, we do not repeat these experiments here but compare TALE directly against SLEB and other training-free pruning methods that represent the strongest existing baselines. Why were methods geared towards tasks so inferior to TALE? TALE is focused on optimizing accuracy on generated output or with LM Eval. Other methods used representational similarity to find redundant layers. So we used a representational similarity technique (cosine similarity or cossim) to guide TALE. TALE registered drops similar to those seen with SLEB and other similarity driven optimization methods. On ARC-Easy, for instance, cossim led TALE to drop 2 layers, **dropping task accuracy** from Llama's baseline of **79.5** to a pruned model accuracy of **58.5** with a time speed up of 1.32. Table 16 in the Appendix provides a direct comparison with SLEB, a similarity based method. We conclude that similarity based optimization for task performance is not competitive with TALE .

As SLEB and other methods use perplexity as a check on pruning, we also used TALE to optimize models for perplexity. TALE provided minimal increases in perplexity with speed up (see Table 15. More importantly, optimizing for perplexity did not translate into better performance on downstream tasks, contra (Song et al., 2024).

5.2 Robustness

We next evaluated TALE on our benchmarks in 0-shot settings across four medium-scale models (LLaMA 3.1 8B, Mistral 7B, Lucie 7B, Qwen 2.5 7B) and one smaller model (Qwen 2.5 0.5B). We wanted to test TALE with respect to improvements in accuracy on actual generated output. All base models could obey this prompt.

Figure 4 compares accuracy of four mid-sized baseline models against pruned counterparts selected using validation splits drawn from the training data; all reported results are evaluated on a disjoint test (or held-out validation) set. Across all benchmarks and base models, Best models yield consistent accuracy gains though percentage gains vary across models and tasks.[1] On Arc Challenge, Llama Best has the lowest increase in accuracy (though still significant) at 1.6%, while Qwen 7b gains 6.3%. Reasoning tasks (Math500, GSM8K) benefited most from TALE with gains ranging from 23% to 51% across all models. Similar results on the larger models using LM-Eval are in the appendix. The fact that we see the same behavior under two different evaluations is evidence that TALE is capturing a real phenomenon, not an artifact of evaluation.

Figure 5 shows that shifts due to different seeds were minimal. Table 2 in the appendix shows numerical values for the averaged results of TALE with 5 different seeds along with variance on both Llama, Qwen, Lucie and Mistral best models. Given the low variance, we take this to show that TALE is robust.

5.3 Computational Cost and Amortized Efficiency

TALE requires a modest, one-time computation to identify the optimal layer-set for a given task, which is then amortized over the model's entire inference lifetime. For an L-layer model and a validation set of size V, the time of pruning process is proportional to $O(I \cdot L \cdot V \cdot T_{\text{layer}})$, where I is the number of pruning iterations (details in Appendix D). For LLaMA 3.1 8B across our benchmarks ($L = 32$, $V \approx 500 - 1500$), the pruning required approximately 1 to 2 GPU-hours on a single A100.

Takeaways. TALE consistently uncovers high accuracy and high accuracy/high efficiency models. By balancing task fidelity with computational savings, it enables both accuracy-focused and efficiency-focused deployment.[2] Even strong, larger models like Qwen 7B see significant improvements, but so do small models (Qwen 0.5B). The stability of the scores under different seeds and different training splits (see Appendix Table 2) underscores the robustness of TALE and the value of its task directed optimization, rather than one-size-fits-all pruning heuristics for model optimization.

[1] Code available at https://anonymous.4open.science/r/tale/

[2] TALE comes with a tunable selection metric for choosing best candidate trade-offs in Appendix J.

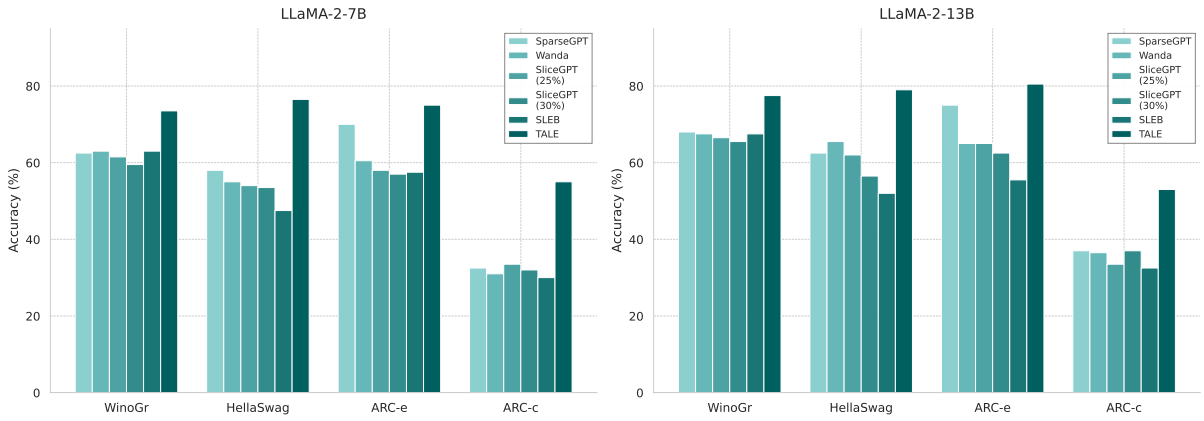


Figure 3: **Performance comparison of pruning methods on LLaMA-2 models.** Evaluation of seven pruning approaches using LM-Eval accuracy across four zero-shot benchmarks for LLaMA-2-7B (left) and LLaMA-2-13B (right). Methods are shown in a blue gradient from light (Baseline) to dark (TALE). TALE achieves the highest accuracy across all tasks while maintaining equivalent sparsity to SLEB, which performed second best to TALE. Method outperformed all structured pruning methods (SpareGPT, Wanda, SliceGPT) on both model sizes, demonstrating effective layer dropping without accuracy loss.

5.4 Interactions between TALE and few shot learning and Fine-tuning

Few-shot setting. As few-shot prompting improves baselines on many tasks,[3] we tested on Lucie and LLaMA models whether TALE could synergize with few shot prompting to bring higher gains. (Appendix Tables 6–7). TALE-pruned variants still achieve higher accuracy in nearly all settings. This shows that TALE-induced improvements are largely complementary to gains from in-context learning.

Fine tuning We also investigated whether TALE's removing- layers before or after fine-tuning affects model performance. Removing layers reduces representational capacity even for Qlora fine-tuing, and so might limit downstream fine-tuning performance compared to baseline instruct-tuned models. Our experiments, however, show the opposite: TALE **not only preserves fine-tuning results but can even improve accuracy and efficiency**.

We explored four settings: (i) fine-tuning the base model (FT), (ii) applying TALE after fine-tuning (FT $\rightarrow$ TALE), (iii) pruning first and then fine-tuning (TALE $\rightarrow$ FT), and (iv) pruning first, then fine-tuning, and finally pruning again (TALE $\rightarrow$ FT $\rightarrow$ TALE). Across various benchmarks, we consistently observed mostly moderate and sometimes significant gains after iterating pruning and fine-tuning, especially on Winogrande and GSM8K (Table 1). This suggests that pruning can act as a regularizer, simplifying the optimization landscape by removing redundant layers.

[3] in particular reasoning tasks like GSM8K and Math500

TALE also reduced computation costs for fine-tuning. For example, pruning LLaMA-3.1 8B before fine-tuning reduced fine-tuning time by 2–2.5 GPU hours on an A100 (an 18.5% reduction) while simultaneously improving Winogrande performance by $+2.4\%$. Iteratively applying pruning and fine-tuning allowed us to prune up to 8 layers achieving still higher accuracy (87.37%) than the full fine-tuned model (85.00%). Similarly, pruning the fully fine-tuned model yielded a 7-layer reduction while maintaining strong accuracy (86.66%).

Overall, these results highlight an unexpected but consistent trend: *pruning with* TALE *does not hinder fine-tuning but instead synergizes with it.* Pruning acts like a regularizer, simplifying the optimization landscape, and can effectively interleave with fine-tuning to create models that are both more accurate and computationally efficient. Pruned models fine-tune faster, require fewer parameters to adapt, and are close to or better in performance than their full counterparts.

6 Discussion

We summarize five key observations below from our experiments.

1. Task dependency of layer importance The literature offers a discussion in the literature about layer importance. Some say early layers are essential (Dalvi et al., 2020); some say late (Tenney et al., 2019; Bansal et al., 2023; Song et al., 2025). Our findings show that layer importance is fundamentally task specific; e.g., removing early layers

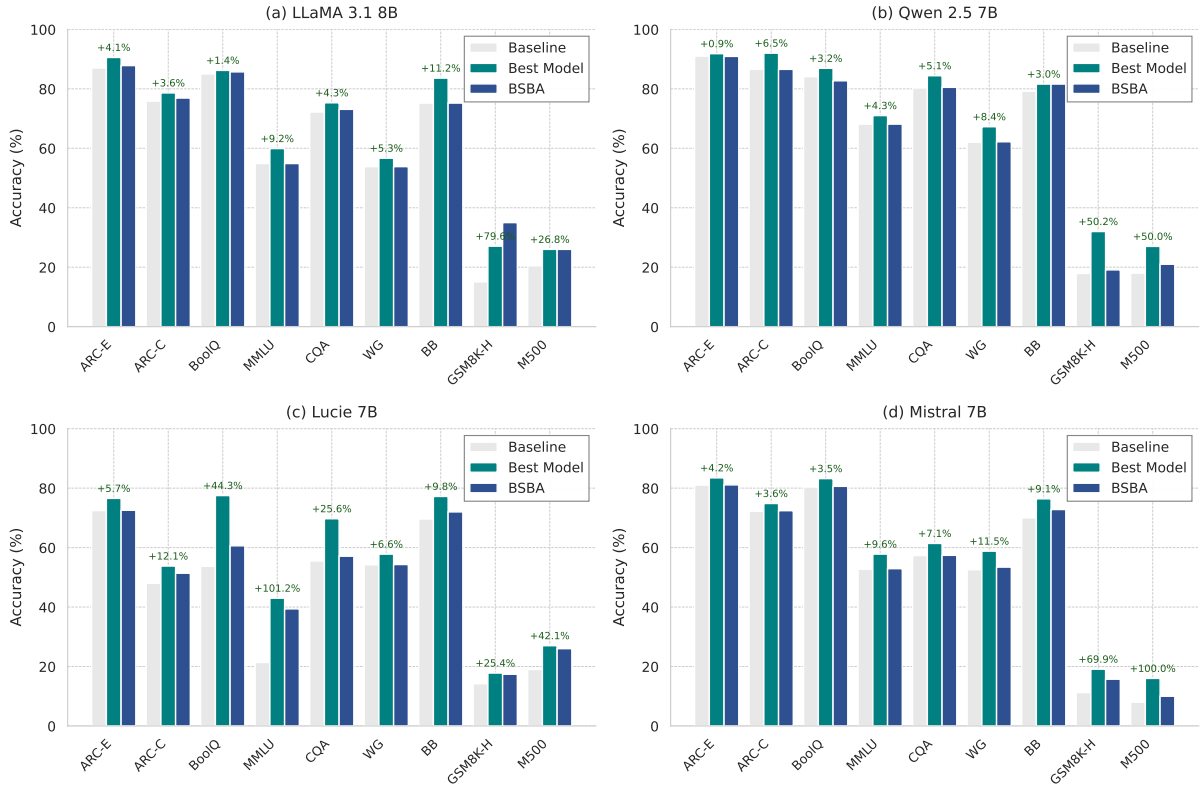


Figure 4: **Zero-shot performance comparison across four language models with layer dropping.** Light blue bars show baseline performance, while dark blue bars represent best model performance after strategic layer dropping. Circled numbers indicate the number of layers dropped (#D) for each task. Results demonstrate that dropping 1–10 layers can improve or maintain performance across most benchmarks, with notable gains on mathematical reasoning tasks (e.g., LLaMA 3.1 8B: 39.0% → 59.0% on GSM8K-HARD with only 1 layer dropped). This suggests significant redundancy in deeper language model architectures.

Model	Dataset	Baseline		TALE		FT Only		TALE → FT		FT → TALE		(TALE → FT) → TALE	
		Perf.	#D	Perf.	#D	Perf.	#D	Perf.	#D	Perf.	#D	Perf.	#D
Llama 3.1 8B	Winogrande	53.83	0	56.67	4	85.00	0	87.06	4	86.74	7	87.37	8
	MMLU	54.87	0	59.90	1	63.62	0	63.49	1	64.21	2	64.01	2
	CommonQA	72.20	0	75.30	3	81.88	0	81.80	3	83.40	3	82.90	6
	GSM8K	15.07	0	37.08	3	42.70	0	53.96	1	50.86	2	54.02	2
Qwen 0.5B	Winogrande	49.86	0	51.88	5	50.43	0	50.43	5	50.49	2	52.49	9
	MMLU	31.48	0	39.98	2	44.87	0	43.76	2	45.53	2	45.58	3

Table 1: Comparison of **Llama 3.1 8B** and **Qwen 0.5B** across Winogrande, MMLU, and CommonQA under different pruning and fine-tuning regimes. Columns denote: (i) Baseline = original model, (ii) Pruned Only = TALE without fine-tuning, (iii) FT Only = fine-tuned without pruning, (iv) Prune → FT = prune then fine-tune, (v) FT → Prune = fine-tune then prune, (vi) (Prune → FT) → Prune = best fine-tuned-pruned model further pruned. Perf. = performance score, #D = number of deleted layers.

reduces accuracy to near zero on commonsense reasoning tasks (Figure 12), but removing LLaMA's layer 3 improves performance on GSM8K-hard.

Related tasks often exhibit similar layer dependencies. Commonsense reasoning tasks (see Figure 12) show importance concentrated in comparable regions of the network. All models showed sizable accuracy boosts in mathematical reasoning tasks after from pruning minimally one to maximally three early to middle layers (e.g., LLaMA layer 3, Mistral layers 6 and 22, Lucie layer 12) (Figures 9, 10, 11). By contrast, knowledge-

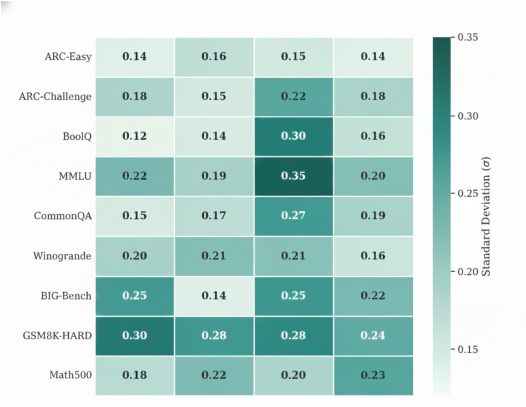


Figure 5: **Model Stability Across Datasets.** Heatmap visualizing the standard deviation (σ) of the final accuracy scores for the Best Model variant of four different Large Language Models (LLMs) across nine datasets. The results are aggregated from five independent runs (seeds). Lighter green indicates higher stability (lower σ); darker green, lower stability (higher σ). LLaMA 3.1 8B shows the overall lowest variance.

intensive tasks ARC, BoolQ, CommonsenseQA, Winogrande, and BIG-Bench) exhibit more modest improvements (though LLaMA boasted an 11% gain on BIG-Bench) and benefit from deleting later layers. These results may help model interpretability, as plotting performance degradation from ablating layers helps localize specific task-solving abilities in the network.

Initial multilingual testing of TALE on Lucie, tuned for French conversational proficiency (Gouvert et al., 2025), with bilingual versions of the same data set showed that optimal pruning was task specific rather than language specific.

Layer redundancy results even with a model trained for a single task. We trained a toy transformer entirely from scratch on a single task: an in-context learning setup involving linear functions. Even in this controlled single-task setting, several layers proved redundant (Figure 6) or degraded performance.

In principle TALE can combine tasks to get more general TALE optimized models. A LLaMA math model without layer 12 improves over baseline LLaMA on Math500 and GSM8K tasks. A promising method to explore is for TALE to prune models on several tasks at once with different mixtures of data to guide the pruning.

2. Model-specific effects with TALE TALE affects different models differently. While Llama benefitted least from TALE , Lucie achieved large gains on MMLU and double-digit gains on ARC-Challenge, CommonsenseQA, BoolQ and GSM8K-hard. TALE conferred more modest but still substantial gains to Qwen-7B and Mistral. Lucie also benefitted from more substantial pruning than the other models. The fact that Lucie was trained on a much smaller dataset (3T tokens vs. 15T for LLaMA and 13T for Qwen) suggests intriguing interactions between pretraining and TALE improvements. We suspect that models trained close to their performance ceiling (via large-scale pretraining, instruction tuning or RLHF) yield smaller gains with TALE , whereas models trained under limited objectives may benefit more.

3. MI Layerwise Analysis. We use Mutual Information (MI) to investigate why selectively removing layers can improve accuracy, focusing on how information about the output evolves as it propagates through the layers. Unlike correlation, MI captures non-linear statistical dependencies and thus provides a more complete measure of dependence (Kinney and Atwal, 2014). We estimate MI using MINE (Belghazi et al., 2018), which is a widely used approximation method. Our analysis reveals that many layers exhibit a pronounced drop in MI (Figure 8). TALE drops some of those layers but not all; overall TALE reduces the peaks and valleys in the graph of MI across layers. However, removing all layers with MI decreases yields very bad performance. Thus, some downward shifts in MI across adjacent layers seems necessary for the model (for more details see Appendix H).

7 Conclusions

TALE removes layers irrelevant to a given task T to consistently yield performance above the base model on T and far above the state of the art in pruning without retraining. TALE also reduces computation costs. It also profitably interacts with further training or fine tuning, further increasing task specific performance. TALE is a generic strategy and can prune at many levels: base pre-trained models, instruction-tuned models, fine-tuned, and post-trained models with RLHF. TALE can benefit high-throughput applications with time constraints (e.g. in multi-agent systems with task-specific agents or interactive AI assistants) and organizations facing trade-offs between model capability and computational costs to use large language models at scale.

Limitations

While TALE demonstrates consistent gains across a broad range of models and tasks, we note several limitations.

First, TALE operates at the level of *entire transformer layers*. This design choice prioritizes simplicity, transparency, and compatibility with existing checkpoints, but it does not exploit finer-grained structure such as attention heads, blocks, or token-level adaptivity. More granular structured pruning or adaptive computation methods may provide complementary benefits, particularly when retraining or architectural modification is permissible.

Second, TALE performs *task-specific* layer selection using a held-out optimization split. As a result, the resulting pruned architectures are specialized to individual tasks rather than universally optimal across tasks. While this specialization aligns with deployment scenarios where the target task is known in advance, it may be less suitable in settings that require a single model to perform well across many heterogeneous tasks without reconfiguration.

Third, TALE relies on a greedy elimination procedure and a stopping tolerance hyperparameter to determine when further layer removal becomes detrimental. Although we observe stable behavior across random seeds and alternative data splits, more principled global optimization strategies or adaptive stopping criteria could further improve robustness and are left for future work.

Finally, our empirical comparisons focus on training-free, layer-level pruning methods that preserve the original model architecture. Approaches that rely on retraining, block-level restructuring, or task-adaptive control flow address a different point in the design space and are therefore not directly comparable within our experimental scope.

Overall, these limitations reflect deliberate design choices rather than deficiencies of the approach. We view TALE as complementary to retraining-based and fine-grained pruning methods, and believe that combining task-aware layer selection with such approaches is a promising direction for future research.

References

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2022. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*.

Guillaume Alain and Yoshua Bengio. 2016. Understanding intermediate layers using linear classifier probes. In *International Conference on Learning Representations (ICLR) Workshop*.

Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. 2016. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*.

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. 2024. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 10865–10873.

Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. 2024. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*.

Hritik Bansal, Karthik Gopalakrishnan, Saket Dingliwal, Sravan Bodapati, Katrin Kirchhoff, and Dan Roth. 2023. Rethinking the role of scale for in-context learning: An interpretability-based case study at 66 billion scale. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11833–11856.

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. 2018. Mutual information neural estimation. In *International conference on machine learning*, pages 531–540. PMLR.

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. 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)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ArXiv:1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Fahim Dalvi, Hassan Sajjad, Nadir Durrani, and Yonatan Belinkov. 2020. Analyzing redundancy

in pretrained transformer models. *arXiv preprint arXiv:2004.04010*.

Wietse De Vries, Andreas Van Cranenburgh, and Malvina Nissim. 2020. What's so special about bert's layers? a closer look at the nlp pipeline in monolingual and multilingual models. *arXiv preprint arXiv:2004.06499*.

Robert M Fano and David Hawkins. 1961. Transmission of information: A statistical theory of communications. *American Journal of Physics*, 29(11):793–794.

Elias Frantar and Dan Alistarh. 2023a. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR.

Elias Frantar and Dan Alistarh. 2023b. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR.

Marylou Gabrié, Andre Manoel, Clément Luneau, Jean Barbier, Nicolas Macris, Florent Krzakala, and Lenka Zdeborová. 2019. Entropy and mutual information in models of deep neural networks. *Advances in Neural Information Processing Systems*, 32.

Shuyang Gao, Greg Ver Steeg, and Aram Galstyan. 2015. Efficient estimation of mutual information for strongly dependent variables. *arXiv preprint arXiv:1411.2003*.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. 2022. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598.

Olivier Gouvert, Julie Hunter, Jérôme Louradour, Christophe Cerisara, Evan Dufraisse, Yaya Sy, Laura Rivière, Jean-Pierre Lorré, and 1 others. 2025. The lucie-7b llm and the lucie training dataset: Open resources for multilingual language generation. *arXiv preprint arXiv:2503.12294*.

Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*. ArXiv:2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Justin B Kinney and Gurinder S Atwal. 2014. Equitability, mutual information, and the maximal information coefficient. *Proceedings of the National Academy of Sciences*, 111(9):3354–3359.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*.

Seungho Park, Seunghan Kim, Jinhyeok Baek, Hoyoung Shin, Minjae Lee, Hyunwook Jang, and Kyungsik Kim. 2024. Gaussian mutual information maximization for efficient graph self-supervised learning. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 8647–8656.

David Peer, Sebastian Stabinger, Stefan Engl, and Antonio Rodríguez-Sánchez. 2022. Greedy-layer pruning: Speeding up transformer models for natural language processing. *Pattern Recognition Letters*, 157:76–82.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Claude E Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Ravid Shwartz-Ziv and Naftali Tishby. 2017. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. 2024. Sleb: Streamlining llms through redundancy verification and elimination of transformer blocks. *arXiv preprint arXiv:2402.09025*.

Xinyuan Song, Keyu Wang, PengXiang Li, Lu Yin, and Shiwei Liu. 2025. Demystifying the roles of llm layers in retrieval, knowledge, and reasoning. *arXiv preprint arXiv:2510.02091*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam R. Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, and 21 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*. Preprint / TMLR.

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*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 4149–4158. Association for Computational Linguistics.

Shengkun Tang, Oliver Sieberling, Eldar Kurtic, Zhiqiang Shen, and Dan Alistarh. 2025. Darwinlm: Evolutionary structured pruning of large language models. *arXiv preprint arXiv:2502.07780*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)*, pages 1–5. Ieee.

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. *arXiv preprint arXiv:1905.09418*.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pechenizkiy, Yi Liang, and 1 others. 2024. Outlier weighed layerwise sparsity (owl) a missing secret sauce for pruning llms to high sparsity. In *Proceedings of the 41st International Conference on Machine Learning*, pages 57101–57115.

Yuxin Zhang, Lirui Zhao, Mingbao Lin, Sun Yunyun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. Dynamic sparse no training: Training-free fine-tuning for sparse llms. In *The Twelfth International Conference on Learning Representations*.

A Implementation Details

Hardware. All experiments were conducted on 1 NVIDIA A100 GPU with 80GB memory.

Models. We applied TALE to five open-weights LLMs of varying scales: **Qwen2.5-0.5B-Instruct**, **Qwen2.5-7B-Instruct**, **Lucie-7B-Instruct**, **Mistral-7B-Instruct**, and **Llama-3.1-8B-Instruct**.

Datasets for TALE pruning. The greedy layer-pruning algorithm was evaluated across nine widely used benchmarks covering reasoning, commonsense, and knowledge-intensive tasks: **ARC-Challenge**, **ARC-Easy**, **MMLU**, **Winogrande**, **GSM8K**, **MATH500**, **CommonQA**, **BIG-Bench**, and **BoolQ**.

Pruning setup. At each iteration, TALE evaluates all candidate single-layer deletions with respect to validation accuracy. The pruning threshold was defined as the baseline accuracy -8% of the full model, ensuring that pruning never reduces performance relative to the original unpruned model. The iterative procedure terminates once no further layer removals satisfy this criterion.

Fine-tuning setup. For fine-tuning experiments, we focused on **Winogrande** and **MMLU**. We employed LoRA with rank 64, a batch size of 4, and the optimizer `paged_adamw_32bit`. A cosine learning rate scheduler was used, and models were trained for 10 epochs.

A.1 Data splits and leakage prevention

For each task, we explicitly separate the data used for layer optimization from the data used for reporting results. Let D denote the full dataset. We partition D into two disjoint subsets: a training set D_{train} and an evaluation set D_{eval}. From D_{train}, we further sample a small held-out subset $D_{\text{opt}} \subset D_{\text{train}}$, which is used exclusively by TALE to guide layer elimination.

At no point does TALE access D_{eval} during optimization. Final accuracy is computed only on D_{eval}. When official test sets are unavailable, D_{eval} corresponds to a held-out validation split that is strictly disjoint from D_{opt}.

Evaluation. The LM-Eval methodology presents a significant limitation: it selects the answer with the highest probability among the provided options rather than assessing what the model would actually generate. This approach ignores hallucination behavior and systematically inflates scores; for example, in a two-choice setting, a hallucinated answer still has a 50% chance of being counted as correct. Furthermore, LM-Eval often assigns relatively high scores to weak models, compressing performance differences and making stronger approaches appear only marginally better despite substantial real-world gains. This produces a misleading picture of model capability, as high LM-Eval results do not guarantee that a model will produce correct, coherent outputs in practice. For these reasons, we relied primarily on Decoder Eval that measures actual accuracy based on the model's generated outputs, which we implemented for each task.

Prompting. For zero-shot and few-shot evaluation, we used task-specific prompts. Below we show the prompt used for datasets, consisting of a system instruction :

ARC-E & ARC-C System Prompt

You are a Science expert assistant. Your task is to answer multiple-choice science questions at grade-school level. Each question has four answer choices, labeled A, B, C, and D.
For each question: - Carefully read the question and all answer choices. - Select the single best answer from the options (A, B, C, or D). - Respond only with the letter of the correct answer, and nothing else—no explanation or extra words.
Be precise and consistent: Only the answer letter.

Bigbench System Prompt

"You are a boolean expression evaluator. You must respond with exactly one word: either 'True' or 'False'. Do not provide explanations, steps, or any other text. Only respond with 'True' or 'False'."

BOOLQ System Prompt

"You are a helpful assistant that answers True/False questions based on given passages. Read the passage carefully and determine if the question can be answered as True or False based on the information in the passage. "Respond with only 'A' for True or 'B' for False."

CommonQA System Prompt

"You are a helpful assistant that answers multiple-choice questions requiring commonsense knowledge and reasoning. Read each question carefully and select the most logical answer from the given options based on common knowledge and reasoning. Respond with only the letter of your chosen answer (A, B, C, D, or E)."

GSM8K System Prompt

"You are a math problem solver. Solve the given math problem step by step. " "Show your complete reasoning and calculations. " "At the end, write your final answer after '####' like this: #### [your final numerical answer]""

MMLU System Prompt

"You are a helpful assistant that answers multiple-choice questions across various academic subjects including humanities, social sciences, STEM, and professional fields. Read each question carefully and select the best answer from the given options. Respond with only the letter of your chosen answer (A, B, C, or D)."

MATH500 System prompt

You are a careful math problem solver. Show complete step-by-step reasoning and all calculations needed to arrive at the answer. Use clear, numbered or labeled steps so the reasoning is easy to follow.
IMPORTANT (formatting):

- After the full reasoning, write the **final answer on a new line by itself** in exactly this format:

  ####
  integer

- `<integer>` must be digits only, optionally with a leading "-" for negatives (e.g., `-7`).
- Do **not** add words, punctuation, units, or commentary on the same line as the #### line.
- The #### line must be the **final line of the output** (nothing may follow it).
- Assume all problems expect integer answers; ensure the final line contains a single integer.

A.2 Layer Removal Implementation

Layer pruning was implemented through a custom model wrapper that reconstructs the architecture excluding specified layers. Given delete indices $\mathcal{D}$, we create a `ModifiedModel` that:

1. Preserves the embedding layer, final normalization, and language modeling head from the original model
2. Constructs a new layer sequence $\mathcal{L}' = \{l_i \mid i \notin \mathcal{D}\}$ by filtering out deleted layers while maintaining the order of retained layers
3. Updates the model configuration to reflect the new layer count

The forward pass implements standard transformer computation: input embeddings are passed through the retained layers sequentially with causal attention masking, then normalized and projected to vocabulary logits. Position embeddings are generated automatically if not provided. This architecture is fully compatible with the Hugging Face training pipeline and can be directly used with LoRA fine-tuning without requiring custom training loops.

A.3 Fine-tuning Training Details

All fine-tuning experiments were conducted using Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA). We employed 4-bit quantization using the BitsAndBytes library to reduce memory footprint during training. The quantization configuration used NF4 (4-bit NormalFloat) quantization type with float16 compute dtype, without nested quantization.

LoRA Configuration: We applied LoRA to all linear layers in the model with the following hyperparameters: rank $r = 64$, alpha $\alpha = 16$, and dropout rate of 0.1. These parameters were kept consistent across both full and pruned models to ensure fair comparison.

Optimization Settings: Training was performed for 10 epochs using the paged AdamW optimizer (32-bit) with a learning rate of 2×10^{-4} and weight decay of 0.001. We used a cosine learning rate schedule with a warmup ratio of 0.03. Gradient clipping was applied with a maximum gradient norm of 0.3. The effective batch size was 60 (per-device batch size of 2 with gradient accumulation steps of 30). Gradient checkpointing was enabled to reduce memory consumption during training.

Data Processing: All sequences were truncated or padded to a maximum length of 300 tokens. We used right-side padding with a special padding token (ID: 128004). Packing was disabled to maintain sequence boundaries, and we removed unused columns from the dataset. The dataloader used 4 workers with the last incomplete batch dropped to ensure consistent batch sizes.

Hardware and Implementation: All experiments were conducted on NVIDIA A100 GPUs. We used mixed-precision training without fp16 or bf16 enabled at the trainer level, relying instead on the 4-bit quantization for memory efficiency. Training logs were reported to TensorBoard every 25 steps.

Dataset	LLaMA 3.1 8B (zero-shot)							Qwen 2.5 7B (zero-shot)						
	Baseline	Best Model			BSBA			Baseline	Best Model			BSBA		
	Perf.	Perf. ± Std	#D	Sp.	Perf.	#D	Sp.	Perf.	Perf. ± Std	#D	Sp.	Perf.	#D	Sp.
ARC-Easy	89	90.4 ± 0.14	5	-14.6%	88.8	8	-23.5%	90.04	93.2 ± 0.16	3	-10.0%	90.08	7	-30.3%
ARC-Challenge	79.4	80.6 ± 0.18	4	-11.7%	77.6	7	-20.5%	86.55	92.00 ± 0.15	2	-6.7%	86.55	6	-19.9%
BoolQ	85.4	85.9 ± 0.12	3	-8.8%	85.4	7	-17.6%	81.90	83.90 ± 0.14	4	-13.3%	82.70	5	-23.2%
MMLU	48.8	53.8 ± 0.22	1	-2.9%	50.2	9	-26.4%	68.10	71.00 ± 0.19	5	-16.6%	68.13	6	-19.9%
CommonQA	72.9	73.8 ± 0.15	3	-8.8%	73.10	6	-17.6%	80.30	84.40 ± 0.17	2	-6.6%	80.50	6	-19.9%
Winogrande	53.8	54.1 ± 0.20	4	-11.7%	53.83	12	-32.2%	62.04	67.25 ± 0.21	3	-10.0%	62.19	6	-19.9%
BIG-Bench	77.2	85.6 ± 0.25	5	-14.4%	76.4	11	-32.2%	79.20	81.60 ± 0.14	6	-19.9%	81.60	6	-19.9%
GSM8K-HARD	39.0	59.0 ± 0.30	1	-2.9%	39.4	4	-11.7%	43.80	61.80 ± 0.28	2	-43.6%	43.99	5	-17.6%
Math500	25.4	28.2 ± 0.18	2	-6.0%	27.4	3	-9.1%	31.00	38.20 ± 0.22	2	-6.6%	32.10	4	-13.3%

Dataset	Lucie 7B (zero-shot)							Mistral 7B (zero-shot)						
	Baseline	Best Model			BSBA			Baseline	Best Model			BSBA		
	Perf.	Perf. ± Std	#D	Sp.	Perf.	#D	Sp.	Perf.	Perf. ± Std	#D	Sp.	Perf.	#D	Sp.
ARC-Easy	74.4	75.8 ± 0.15	6	-18.1%	73.8	8	-23.5%	83.8	85.6 ± 0.14	5	-15.4%	82.8	9	-27.7%
ARC-Challenge	46.0	51.45 ± 0.22	7	-22.1%	48.8	11	-33.1%	76.2	79.1 ± 0.18	6	-18.5%	76.2	8	-24.6%
BoolQ	53.0	74.0 ± 0.30	5	-17.2%	63.0	19	-54.2%	81.3	84.4 ± 0.16	4	-18.5%	80.8	5	-27.7%
MMLU	13.0	54.0 ± 0.35	8	-24.1%	15	22	-60.2%	39.4	40.8 ± 0.20	2	-6.2%	39.0	8	-24.6%
CommonQA	54.2	68.6 ± 0.27	3	-9.1%	54.6	17	-48.2%	61.0	64.4 ± 0.19	4	-12.3%	61.6	7	-21.5%
Winogrande	51.6	53.1 ± 0.21	5	-27.1%	53.0	15	-45.2%	53.2	54.3 ± 0.16	10	-30.7%	52.4	13	-40.0%
BIG-Bench	67.4	75.0 ± 0.25	9	-27.1%	71	15	-45.1%	70.4	75.4 ± 0.22	9	-28.0%	72.6	11	-33.8%
GSM8K-HARD	32	39.0 ± 0.28	1	-3.1%	37	3	-9.1%	24	33 ± 0.24	2	-6.2%	26.1	4	-12.3%
Math500	21.0	26.1 ± 0.20	2	-6.0%	25.1	3	-9.1%	19	28 ± 0.23	1	-3.1%	18.8	4	-12.3%

Table 2: Robustness study of the proposed layer-dropping method across multiple language models under zero-shot evaluation. For each dataset and model, results are reported over five random seeds to account for variability in decoding and sampling. We present the baseline model accuracy and the accuracy of the best pruned configuration, along with their corresponding standard deviations computed across the 5 seeds. The table also includes the number of transformer layers removed in the best-performing configuration (**#D**) and the resulting inference speedup (**Sp.**) expressed as the percentage of total TFlops saved during evaluation. All experiments use 10% of the training split for optimization and evaluate on the respective test sets. Bold values indicate the highest mean accuracy for each dataset.

B Robustness Study on TALE

See Tables 2 and 17.

C Ablation study on validation Set of Pruning

We analyze the effect of validation set size on TALE's layer selection. Table 3 reports the specific layers dropped for different validation set sizes across three tasks (ARC-Easy, MMLU, GSM8K) and two models (Llama 3.1 8B, Qwen 2.5 7B).

Model	Val Size	Task	Dropped Layers
Llama 3.1 8B	200	ARC-E	{19, 20, 22, 29, 32 }
		MMLU	{ 21 }
		GSM8K	{ 3 }
	500	ARC-E	{ 19, 20, 21, 29, 32 }
		MMLU	{ 21 }
		GSM8K	{ 3 }
	1000	ARC-E	{ 19, 20, 21, 29, 32 }
		MMLU	{ 21 }
		GSM8K	{ 3 }
Qwen 2.5 7B	100	ARC-E	{ 22 , 27 , 28 }
		MMLU	{ 18 , 22 , 24 , 27 , 28 }
		GSM8K	{ 19 }
	500	ARC-E	{ 19, 22 , 28 }
		MMLU	{ 22 , 23 , 26 , 27 , 28 }
		GSM8K	{ 19 }
	1000	ARC-E	{ 19 , 22 , 28 }
		MMLU	{ 22 , 23 , 26 , 27 , 28 }
		GSM8K	{ 19 }

Table 3: Layers removed by TALE for different validation-set sizes across three tasks. This reveals the stability of pruning decisions directly.

D Number of parameters per layer for each model

Model	Params/Layer	Layers
LLaMA 3.1 8B	218,112,000	32
Qwen 2.5 7B	233,057,792	28
Mistral 7B	218,112,000	32
Lucie 7B	192,946,176	32
Qwen 2.5 0.5B	14,912,384	24

Table 4: Model parameter counts comparison showing parameters per layer and total number of layers.

E Practical computing savings and scaling

We quantify TALE's inference-cost reduction by measuring TFLOPs (tera-FLOPs) drop per removed layer. Across models and tasks, removing a single transformer layer yields a mean TFLOPs reduction of $3.00\% \pm 0.20\%$. Because TALE removes entire layers sequentially, the total TFLOPs reduction scales essentially linearly with the number of iterations (layers removed). In practice this means only a few iterations are required to reach common sparsity targets: e.g., three iterations remove roughly $\approx 9\%$ TFLOPs, sufficient to realize $\approx 10\%$ sparsity in our settings.

Wall-clock runtime and end-to-end cost. In addition to FLOPs-based estimates, we report wall-clock runtime measurements under the exact decoding settings used for evaluation. TALE incurs a one-time optimization cost per task, after which the pruned model is used for standard inference with reduced depth.

For all non-reasoning benchmarks (e.g., ARC, BoolQ, MMLU, CommonsenseQA), we use greedy decoding with `max_new_tokens = 1`, reflecting single-step answer generation. Under this setting, the total wall-clock time required to complete a full TALE optimization run is approximately 1 hour per dataset on a single NVIDIA A100 GPU.

For reasoning-heavy benchmarks (GSM8K-Hard and Math500), we use `max_new_tokens = 200` to allow full chain-of-thought generation. Due to longer decoding sequences, the average wall-clock time for TALE on these datasets is approximately 3 hours, measured over the evaluated subset.

Importantly, this optimization cost is incurred only once per task. During deployment, inference is performed using the pruned model, yielding reduced per-example latency and throughput improvements proportional to the number of eliminated layers. These gains are amortized over all subsequent inference calls.

F Layer Redundancy Is Not a Multi-Task Artifact

F.1 Training details

We train a transformer model to perform in-context learning over the class of linear functions. Given a prompt consisting of a small set of input–output examples $(x_1, g(x_1), \ldots, x_p, g(x_p), x)$, the model is tasked with predicting the value $g(x)$ for the final input x.

Our architecture is a 12-layer transformer with 8 attention heads, trained **from scratch** exclusively on the linear function family. The training objective follows the standard ICL formulation introduced in (Garg et al., 2022; Akyürek et al., 2022):

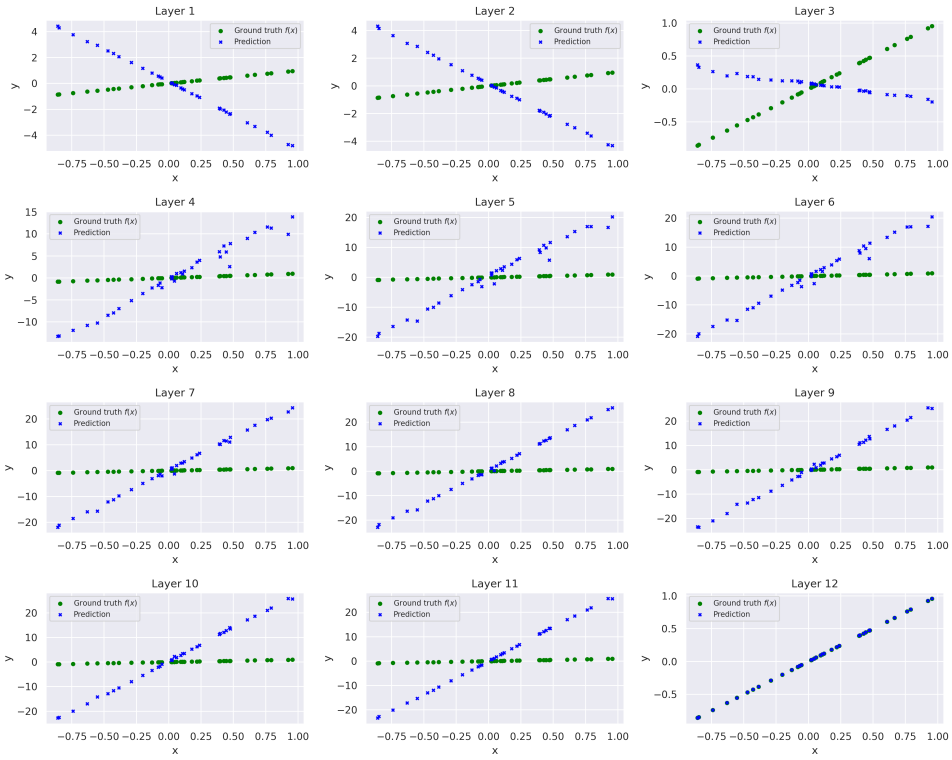


Figure 6: Plots showing evolution of the predictions over layers for $f(x) = x$ for a model trained on degree 1 with $D_{\mathcal{I}}, D_{\mathcal{F}} \sim \mathcal{U}(-1, 1)$.

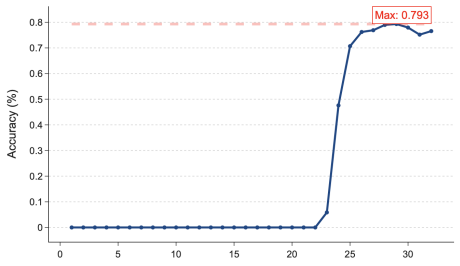
$$\min_{\theta} \mathbb{E}_{g\sim\mathcal{D}_{\mathcal{F}}}\Bigg[\mathbb{E}_{x_1,\ldots,x_p\sim\mathcal{D}_{\mathcal{I}}} \sum_{i=0}^{k} \ell\Big(y_{i+1}, f^{\theta}\big(x_1, g(x_1), \ldots, x_{i+1}\big)\Big)\Bigg] \tag{1}$$

where ℓ is the squared error loss. Here, $y_{i+1} = g(x_{i+1})$ denotes the ground-truth output of the underlying linear function g.

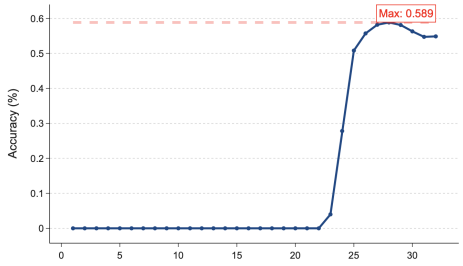
F.2 Plots

Figure 6

G Intution behind TALE



(a) ARC-Challenge



(b) MMLU

Figure 7: Layer-wise output performance for LLaMA models: results when generating predictions from intermediate layers 1 through 32 on two different datasets.

H Information theory: Why pruned models might perform better.

Our results pose a puzzle: the increase in accuracy with TALE is counterintuitive: why would removing parts of a carefully trained model lead to better performance? One way to explore this question is mutual information.

(Alemi et al., 2016; Tishby and Zaslavsky, 2015) use information theory (Shannon, 1948) to analyze how neural networks learn and represent data. (Fano and Hawkins, 1961) define $\mathrm{I(X;Y)}$, the mutual information between two random variables X and Y, with the equation:

$$\begin{aligned} \mathrm{I(X;Y)} &= \mathrm{H(Y)} - \mathrm{H(Y \mid X)} \\ &= \mathrm{H(X)} - \mathrm{H(X \mid Y)} \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)\,p(y)} \end{aligned} \tag{2}$$

where $p(x,y)$ is the joint distribution of X and Y, and $p(x), p(y)$ are their marginals and where $\mathrm{H(X)} = -\sum_x p(x) \log p(x)$ is the (Shannon, 1948) entropy. $\mathrm{I(X;Y)}$ measures how much knowing X reduces uncertainty about Y (Tishby and Zaslavsky, 2015; Shwartz-Ziv and Tishby, 2017). To attempt to explain why accuracy increases through task pruning we also use MI.

A major challenge of this approach is that it requires information about true distributions, which are infeasible to compute. As a result, researchers typically assume a Gaussian distribution (Gabrié et al., 2019; Gao et al., 2015; Park et al., 2024) or approximate the probe using a classifier (Belinkov, 2022; Alain and Bengio, 2016) or an MLP (Belghazi et al., 2018). These approximations can yield useful insights. In our case, the Gaussian assumption did not fit our datasets. Since we evaluate on QA tasks, we used a trainable classifier to approximate the probes and estimate $I(X^\ell, \mathrm{Y})$ at each layer, where X^ℓ denotes the contextualized representations at layer ℓ and Y denotes the target answer. This approximates how much information the layer ℓ representations contain about the answer. The goal is then to examine whether some layers exhibit a sharp drop in information and whether those layers coincide with the ones whose removal leads to improved performance.

Our findings, summarized in Figure 9 and Table 9, reveal two key patterns: (i) several layers in large pre-trained transformers exhibit a pronounced drop in mutual information; (ii) removing layers dictated by TALE consistently increases the mutual information at the subsequent layer across tasks. Together, these results suggest that certain layers act more as bottlenecks than as contributors to task-relevant representations, providing a rationale for why pruning can lead to improved accuracy.

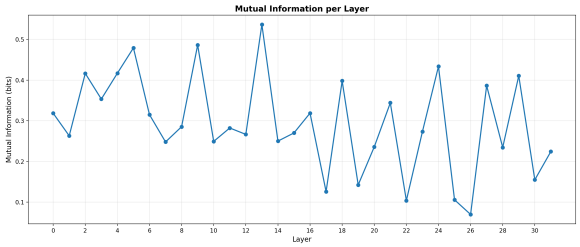
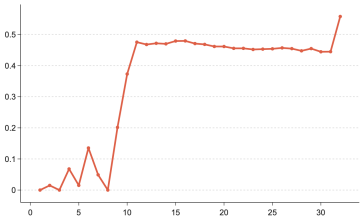


Figure 8: Evolution of Mutual Information about the output through layers for Llama3-8B.

(a) ARC-Easy (Qwen 0.5B)



(b) BoolQ (Lucie 7B)

(c) BigBench (Llama 8B)

Figure 9: Evolution of mutual information (MI) across transformer layers for different benchmark datasets and different models. Each subplot shows how information is processed and transformed as it flows through the network layers, demonstrating distinct patterns of information propagation for (a) ARC-Easy on Qwen 0.5B, (b) BoolQ on Lucie 7B, and (c) BigBench on LLaMA 8B.

I LM-Eval as a sanity check

LM-Eval is a commonly used multiple-choice evaluation protocol and is included here solely as a sanity check for comparability with prior work. The LM-Eval results reported in Table 5 are obtained by evaluating the same pruned models selected using the optimization protocol described in Appendix A.1.

Dataset	LLaMA 3.1 8B 0-shots						
	Baseline	Best Model			BSBA		
	Perf.	Perf.	#D	Sp.	Perf.	#D	Sp.
BoolQ	82.4	87.63	3	-	85.62	7	-
Hellaswag	52.5	55.5	3	-	54.5	5	-
COMMONQA	77.2	81.61	6	-	80.27	7	-
WINOGRANDE	75.92	78.93	4	-	76.59	5	-

Table 5: Results of **LLaMA 3.1 8B** across nine benchmarks. All tested on 0-shot and evaluated with lm eval

J A tunable metric for finding accuracy vs. speed up optimization

To systematically select among these candidates according to user priorities, we propose the Accuracy–Efficiency Harmonic Mean (AE-HM):

$$\begin{aligned} r_A &= \frac{\text{Acc(Model)}}{\text{Acc(Baseline)}}, \\ \text{AE-HM(Model)} &= \frac{(1+\lambda^2) r_A \, S}{\lambda^2 S + r_A} \\ &= \frac{1+\lambda^2}{\frac{\lambda^2}{r_A} + \frac{1}{S}} \end{aligned} \tag{3}$$

where S denotes the relative inference speedup and λ controls the relative importance of accuracy versus efficiency. The user can set AE-HE's parameter λ to desired specifications: if $\lambda > 1$, we prioritize r_A; if $\lambda < 1$ we prioritize Speedup.

By computing AE-HM for candidate models, we can automatically identify the model with the highest score for a given task or a set of tasks given a particular AE-HM parameter setting:

$$M_{\text{best-compromise}} = \arg\max_i \text{AE-HM}(M_i) \tag{4}$$

Figure 10: Relative Gain comparison across datasets. LLaMA $\beta = 3$

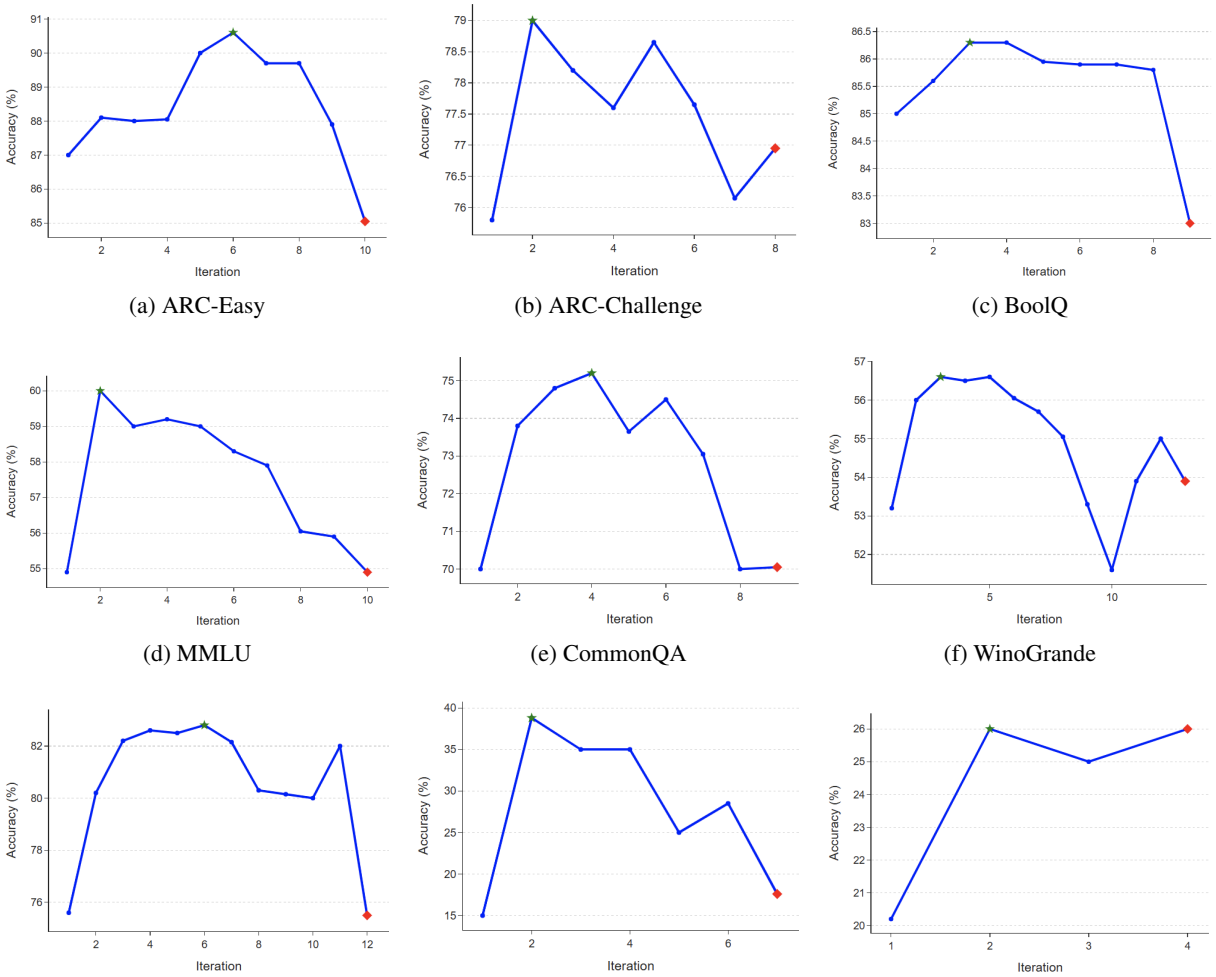
K Results

Dataset	Lucie 7B few-shots						
	Baseline	Best Model			BSBA		
	Perf.	Perf.	#D	Sp.	Perf.	#D	Sp.
ARC-Easy	69.2	72.36	9	1.41	71.27	12	1.68
ARC-Challenge	49.31	55,17	9	1.39	51.72	13	1.67
BoolQ	77.6	79.10	6	1.22	78.5	10	1.27
MMLU	41.02	43.44	7	1.26	41.48	11	1.55
COMMONQA	55.4	69.7	3	1.22	57.10	17	2.02
WINOGRANDE	52.8	56.90	12	1.58	53.30	17	1.74
BIG-Bench	68.8	77.20	9	1.61	72	15	2.23
GSM8K-HARD	26.97	29.21	1	1.03	26.97	2	1.1

Table 6: Results of **Lucie 7B** across nine benchmarks. All tested on 5-shots, except gms8k on 8-shots Performance (%) cells are color-coded: green = gain, red = decline, and gray = near-neutral change compared to baseline.

Dataset	LLaMA 3.1 8B few-shots						
	Baseline	Best Model			BSBA		
	Perf.	Perf.	#D	Sp.	Perf.	#D	Sp.
ARC-Easy	90.36	**92.18** 2.01% ↑	4	1.14	90.91	8	1.37
ARC-Challenge	78.2	**83.10** 6.27% ↑	3	1.17	78.62	9	1.42
BoolQ	82.7	85.3 3.1% ↑	4	1.11	83.0	6	1.22
MMLU	59.2	62.38 5.37% ↑	4	1.14	59.57	7	1.26
COMMONQA	73.30	75.30 2.72% ↑	6	1.22	73.80	7	1.32
WINOGRANDE	57.01	60.1 5,26% ↑	3	1.1	57.02	8	1.3
BIG-Bench	70.0	83.60 19,43% ↑	5	1.2	81.20	15	1.83
GSM8K-HARD	60.67	60.67	0	1	60.67	0	1
MATH500	44.00	49.00 11.36% ↑	1	1.02	45.00	2	1.03

Table 7: Results of **LLaMA 3.1 8B** across nine benchmarks. All tested on 5-shots, except gms8k and MATH500 on 8-shots



(a) ARC-Easy (b) ARC-Challenge (c) BoolQ

(d) MMLU (e) CommonQA (f) WinoGrande

(g) BIG-Bench (h) GSM8K-Hard (i) Math500

Figure 11: Accuracy progression of TALE across 9 benchmark datasets for LLaMA 3.1 8B. Each curve represents the accuracy at successive iterations. The ⋆ denotes the best-performing layer drop configuration, while the □ highlights the Best Speed up with at least Baseline Accuracy (BSBA) configuration.

L Deleted Layers in each Model and Benchmark

Dataset	Best Model	BSBA
ARC-Easy	19 25 27 28	19 20 21 24 25 26 27 28
ARC-Challenge	19 22 27	19 20 21 22 23 24 26 27 28
BoolQ	19 25 26 32	15 19 21 22 25 26 30 32
MMLU	20 21 27 28	20 21 22 24 27 28 32
CommonQA	21 22 27 28 31 32	21 22 23 27 28 31 32
Winogrande	20 22 24	17 19 20 22 24 26 29 32
BIG-Bench	11 16 20 21 26	10 11 16 20 21 22 23 24 26 27 28 29 30 31 32
MATH500	28	24 28

Table 8: Deleted layers represented as color-coded inline numbers. Blue = Best Model, Orange = BSBA for LlaMA 3.1 8B with few-shots.

Table 9 shows how using AE-HM allows us to bring model size down effectively on our BSBA Llama model with 0 shot performance on our nine data sets. The BSBA LLama model had speed up gains between 27 and 46% on our various benchmarks and maintained performance at or above original model levels (See Table 9).

Dataset	Best Model	BSBA
ARC-Easy	19 20 21 29 32	19 20 21 22 25 27 29 32
ARC-Challenge	19 20 23 27	19 20 21 23 25 27 28
BoolQ	21 23 28	18 21 22 27 28 32
MMLU	21	19 21 22 24 25 26 27 28 31
CommonQA	19 23 28	19 22 23 26 27 28
Winogrande	23 24 26 32	20 21 22 23 24 25 26 27 29 31 32
BIG-Bench	14 20 22 28 29	14 18 20 21 22 23 24 28 29 31 32
GSM8K-Hard	3	3 21 22 25 26 27 29

Table 9: Deleted layers represented as color-coded inline numbers. Blue = Best Model, Orange = BSBA for LlaMA 3.1 8B 0 shot.

Dataset	Best Model	BSBA
ARC-Easy	19 22 28	6 19 22 24 26 27 28
ARC-Challenge	27 28	7 22 23 26 27 28
BoolQ	18 21 27 28	12 19 21 22 26 27 28
MMLU	22 23 26 27 28	18 22 23 26 27 28
CommonQA	22 28	6 21 22 23 27 28
Winogrande	22 26 27	6 20 22 25 26 27
BIG-Bench	10 19 23 25 26 27	10 19 23 25 26 27

Table 10: Deleted layers represented as color-coded inline numbers. Blue = Best Model, Orange = BSBA for **Qwen 2.5 7B** zero-shot.

Dataset	Best Model	BSBA
ARC-Easy	15 16 23 24 27 28	13 15 16 18 19 20 21 22 23 24 25 27 28
ARC-Challenge	16 18 20 21 23 25 26	15 16 18 19 20 21 22 23 25 26 28
BoolQ	8 17 25 28 29	5 8 11 12 13 14 15 16 17 19 20 23 25 26 27 28 29 31
MMLU	11 12 15 16 20 21 22 28	5 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 28 30 31
CommonQA	11 12 27	11 12 13 15 16 17 18 19 20 21 22 23 24 25 27 28
BIG-Bench	6 7 15 17 20 21 25 26 27	6 7 13 15 17 19 20 21 22 24 25 26 27 28 29
GSM8K-Hard	12	12 21 23

Table 11: Deleted layers represented as color-coded inline numbers. Blue = Best Model, Orange = BSBA for Lucie 7B 0 shots.

Dataset	Best Model	BSBA
ARC-Easy	21 22 24 26 29	21 22 23 24 25 26 29 30 32
ARC-Challenge	22 24 25 27 28 30	21 22 24 25 26 27 28 30
BoolQ	17 22 23 24 27 32	12 17 21 23 24 25 27 28 32
MMLU	24 30	22 23 24 25 26 27 30 32
CommonQA	19 22 25 28	19 21 22 24 25 28 32
Winogrande	18 19 20 22 23 24 26 27 31 32	4 13 18 19 20 22 23 24 26 27 29 31 32
BIG-Bench	3 5 15 22 23 24 26 27 28	3 5 14 15 18 22 23 24 26 27 28
GSM8K-Hard	6 22	6 11 22 28

Table 12: Deleted layers represented as color-ccdinline numbers. Blue = Best Model, Orange = BSBA for **Mistral** zero-shot.

Dataset / Qwen 2.5 7B	LLaMA 3.1 8B									
	Baseline	Best Model			Baseline	Best Model				
Perf.	Perf #D	Perf. Sp.	#D Perf.	Sp. #D	Perf. Sp.	#D	Sp.	Perf.	#D	Sp.
ARC-Easy	87.00	**90.55** (+3.55% ↑)	5	1.27	85.09 (-1.91% ↓)	**9**	**1.48**	90.04	**94.40**	3
1.17	90.08	7	1.5							
ARC-Challenge	75.86	**78.62** (+2.76% ↑)	4	1.26	76.90 (+1.04% ↑)	**7**	**1.41**	86.55	**92.00** (+5.45% ↑)	2
1.3	86.55	6	1.37							
BoolQ	85.00	**86.20** (+1.20% ↑)	3	1.10	83.00 (-2.00% ↓)	**8**	**1.36**	81.9		83.9
4	1.3	80.00	7	1.45						
MMLU	54.87	**59.90** (+5.03% ↑)	1	1.05	54.87	**9**	**1.37**	68.10	71.00	5
1.24	68.13	6	1.31							
COMMONQA	72.20	0	1.00	**75.30** (+3.10% ↑)	3	1.21	70.10 (-2.10% ↓)	**8**	**1.34**	80.30
84.4	2	1.12	80.50	6	1.27					
WINOGRANDE	53.83	0	1.00	**56.67** (+2.84% ↑)	4	1.25	53.83	**12**	**1.47**	62.04
0	1.00	67.25	3	1.16	62.19	6	1.48			
BIG-Bench	75.20	0	1.00	**83.60** (+8.40% ↑)	5	1.24	75.20	11	1.58	79.2
81.60	6	1.47	81.60	6	1.47					
GSM8K-HARD	15.07	0	1.00	**38.02** (+22.95% ↑)	1	1.12	17.31 (+2.24% ↑)	**6**	**1.35**	-
-	-	-	-	-	-	-	-			
GSM8K*	42.15	0	1.00	**56.56** (+14.41% ↑)	1	–	48.14 (+5.99% ↑)	3	–	-
-	-	-	-	-	-	-	-			

Table 13: Comparison of **LLaMA 3.1 8B** and **Qwen 2.5 7B** across nine benchmarks. We report accuracy (%), number of layers dropped, and relative inference speedup. Best accuracy per dataset is in **bold**. (+X% ↑) shows accuracy improvement from baseline, (-X% ↓) shows accuracy decline.

M More on pruning and a common pruned layers model

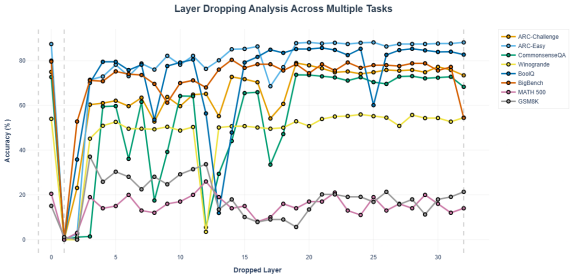


Figure 12: Nine benchmark tasks indicating performance after one layer is dropped from different positions in Llama3-8B.

N General Pruning results

Group	Dataset	Baseline	Pruned Model	speedup
Common-sense	ARC-Easy	87.0	87.82	1.2
	ARC-Challenge	75.86	75.00	1.21
	CommonQA	72.20	64.70	1.1
	Winogrande	54.20	50.57	1.13
Reading	BoolQ	85.0	85.5	1.17
	BIG-Bench	75.2	67.2	1.1

Table 14: Accuracy of LLaMA-3.1-8B (baseline) versus a pruned variant obtained by dropping layers selected through BSBA. For each task, BSBA identified removable layers, and we retained the intersection of layers that appeared in at least 75% of tasks within the Common-sense group (layers 19, 22, 23, 27) and (layers 18, 21, 22, 28, 32) for Reading Comprehension tasks. These layers were then pruned globally from the model, and performance was re-evaluated across tasks. Speedup is reported relative to the baseline.

O TALE evaluation with perplexity

Model	WikiText2		LAMBADA	
	Vanilla	TALE	Vanilla	TALE
LLaMA 3.1 8B	24.6	24.9	28.1	28.9
Lucie 7B	46.1	36.4	52.5	43.8

Table 15: Perplexity scores for two models across WikiText2 and LAMBADA with Vanilla and TALE (sparisty 10%) configurations.

Model	Method	Sparsity	WinoGr	ARC-e	ARC-c
LLaMA-2-7B	Baseline	0%	41.2	51.7	40
	SLEB	10%	18 (-56.3% ↓)	29 (-43.9% ↓)	28.8 (-28.0% ↓)
	TALE	10%	**56** (+35.9% ↑)	**62.3** (+20.5% ↑)	**50** (+25.0% ↑)
	TALE	25%	**51** (+23.8% ↑)	**64.8** (+25.3% ↑)	**47.6** (+19.0% ↑)
LLaMA-2-13B	Baseline	0%	42	73.0	54.9
	SLEB	10%	24.2 (-42.3% ↓)	43.5 (-40.4% ↓)	29.8 (-47.3% ↓)
	TALE	10%	**56.4** (+34.3% ↑)	**77.3** (+5.9% ↑)	**64.4** (+17.1% ↑)
	TALE	25%	**55.2** (+31.4% ↑)	**75.3** (+3.2% ↑)	**64.1** (+16.4% ↑)

Table 16: Accuracies (%) with Decoder Eval on zero-shot tasks for LLaMA-2-7B and LLaMA-2-13B

P Scores with relative increases in accuracy

Dataset	LLaMA 3.1 8B (zero-shot)							Qwen 2.5 7B (zero-shot)						
	Baseline	Best Model			BSBA			Baseline	Best Model			BSBA		
	Perf.	Perf.	#D	Sp.	Perf.	#D	Sp.	Perf.	Perf.	#D	Sp. saved	Perf.	#D	Sp.
ARC-Easy	87.00	**90.55**(+4.08% ↑)	5	-14.6%	87.82	8	-23.5%	91.01	**91.82**(+0.89% ↑)	2	-10.0%	90.91	5	-30.3%
ARC-Challenge	75.86	**78.62**(+3.63% ↑)	4	-11.7%	76.90	7	-20.5%	86.55	**92.00**(+6.45% ↑)	2	-6.7%	86.55	6	-19.9%
BoolQ	85.00	**86.20**(+1.40% ↑)	3	-8.8%	85.70	7	-17.6%	84.10	**86.90**(+3.22% ↑)	4	-13.3%	82.70	5	-23.2%
MMLU	54.87	**59.90**(+9.17% ↑)	1	-2.9%	54.87	9	-26.4%	68.10	**71.00**(+4.26% ↑)	5	-16.6%	68.13	6	-19.9%
CommonQA	72.20	**75.30**(+4.29% ↑)	3	-8.8%	73.10	6	-17.6%	80.30	**84.40**(+5.11% ↑)	2	-6.6%	80.50	6	-19.9%
Winogrande	53.83	**56.67**(+5.28% ↑)	4	-11.7%	53.83	12	-32.2%	62.04	**67.25**(+8.40% ↑)	3	-10.0%	62.19	6	-19.9%
BIG-Bench	75.20	**83.60**(+11.17% ↑)	5	-14.4%	75.20	11	-32.2%	79.20	**81.60**(+3.03% ↑)	6	-19.9%	81.60	6	-19.9%
GSM8K-HARD	15.07	**37.08**(+146.05% ↑)	1	-2.9%	35.0	4	-11.7%	7.9	**27.0**(+243.58% ↑)	2	-6.6%	19.1	4	-13.3%
Math500	20.50	**26.00**(+26.83% ↑)	1	-2.9%	26.00	3	-8.8%	18.00	**27.00**(+50.0% ↑)	2	-6.6%	21.00	4	-13.3%

Dataset	Lucie 7B (zero-shot)							Mistral 7B (zero-shot)						
	Baseline	Best Model			BSBA			Baseline	Best Model			BSBA		
	Perf.	Perf.	#D	Sp.	Perf.	#D	Sp.	Perf.	Perf.	#D	Sp.	Perf.	#D	Sp.
ARC-Easy	72.45	**76.55**(+5.66% ↑)	6	-18.1%	72.55	13	-39.2%	81.02	**83.45**(+4.23% ↑)	5	-15.4%	81.09	9	-27.7%
ARC-Challenge	48.00	**53.79**(+12.06% ↑)	7	-21.1%	51.38	11	-33.1%	72.20	**74.83**(+3.64% ↑)	6	-18.5%	72.41	8	-24.6%
BoolQ	53.70	**77.50**(+44.32% ↑)	5	-17.2%	60.60	19	-54.2%	80.36	**83.20**(+3.53% ↑)	6	-18.5%	80.60	10	-27.7%
MMLU	21.36	**42.98**(+101.2% ↑)	8	-24.1%	39.39	15	-45.2%	52.73	**57.81**(+9.63% ↑)	2	-6.2%	52.91	8	-24.6%
CommonQA	55.50	**69.70**(+25.59% ↑)	3	-9.1%	57.10	17	-48.2%	57.32	**61.40**(+7.12% ↑)	4	-12.3%	57.40	7	-21.5%
Winogrande	54.20	**57.80**(+6.64% ↑)	5	-27.1%	54.30	15	-45.2%	52.55	**58.80**(+11.53% ↑)	10	-30.7%	53.43	13	-40.0%
BIG-Bench	69.60	**77.20**(+9.84% ↑)	9	-27.1%	72.00	15	-45.1%	70.00	**76.40**(+9.14% ↑)	9	-28.0%	72.80	11	-33.8%
GSM8K-HARD	14.20	**17.80**(+25.35% ↑)	1	-3.1%	17.40	3	-9.1%	11.24	**19.10**(+69.92% ↑)	2	-6.2%	15.73	4	-12.3%
Math500	19.00	**27.00**(+42.11% ↑)	2	-6.0%	26.00	3	-9.1%	8.00	**16.00**(+100% ↑)	1	-3.1%	10.00	4	-12.3%

Dataset	Qwen 2.5 0.5B (zero-shot)						
	Baseline	Best Model			BSBA		
	Perf.	Perf.	#D	Sp.	Perf.	#D	Sp.
ARC-Easy	40.00	**60.91**(+48.49% ↑)	3	1.1	48.36	5	1.4
ARC-Challenge	35.52	**40.34**(+13.57% ↑)	1	1.1	37.24	4	1.5
BoolQ	62.30	**67.20**(+7.87% ↑)	5	1.4	66.20	6	1.5
MMLU	31.48	**39.97**(+26.96% ↑)	2	1.1	33.90	5	1.4
CommonQA	42.40	**49.10**(+15.80% ↑)	2	1.3	44.00	3	1.4
Winogrande	49.86	**51.88**(+4.51% ↑)	5	1.3	49.87	17	3.9
BIG-Bench	72.40	**73.60**(+1.66% ↑)	2	1.2	73.60	2	1.2
GSM8K-HARD	6.74	**11.24**(+66.77% ↑)	1	1.2	8.99	2	1.2
Math500	8.00	**12.00**(+50.00% ↑)	1	1.1	9.00	2	1.1

Table 17: Performance comparison for LLaMA 3.1 8B, Qwen 2.5 7B, Lucie 7b, Mistral 7b and Qwen 2.5 0.5B under 0-shot evaluation on the 9 benchmarks with the training test split as in Table 2). We report accuracy (%), number of layers dropped, and relative inference speed in time.