Hopscotch: Discovering and Skipping Redundancies in Language Models

Mustafa Eyceoz[1], Nikhil Shivakumar Nayak[1], Hao Wang[1], Ligong Han[1], Akash Srivastava[1]

[1]Red Hat AI Innovation

Correspondence: meyceoz@redhat.com

Abstract

Modern causal language models stack many attention blocks to improve performance, but not all blocks are necessary for every task. We propose Hopscotch, a simple yet effective method that identifies and skips attention blocks with least contributions to a task and adapts to preserve output quality. Hopscotch jointly optimizes which blocks to skip and how to scale the outputs of the remaining layers. By introducing lightweight, trainable scaling parameters to attention and MLP blocks, it mitigates distribution shifts in hidden states caused by removing attention blocks. Hopscotch does not modify model weights or require access to pretraining or instruction-tuning data, and is compatible with existing model compression techniques. When applied to `Llama-3.1-8B` and `Qwen2.5-7B`, Hopscotch achieves less than a 2% drop in performance even after skipping four attention blocks.

1 Introduction

Large language models (LLMs) continue to grow in size, driven by "scaling laws" that suggest larger models tend to yield better performance (Hestness et al., 2017; Hoffmann et al., 2022; Henighan et al., 2020). Adding more attention blocks increases model capacity, but self-attention is also the most expensive operation in LLMs. Unlike MLP blocks, attention incurs a quadratic computational cost with respect to sequence length, making it a dominant factor in inference-time efficiency. However, not all attention blocks are equally important for every task, and some may carry redundant information. In this paper, we explore whether entire attention blocks can be skipped without significant performance degradation.

We introduce **Hopscotch**[1], a method that jointly learns which attention blocks to skip and how to rescale the outputs of the remaining attention and MLP blocks. Hopscotch iteratively identifies attention blocks with minimal contribution to the target task and introduces lightweight, trainable scaling parameters to mitigate distribution shifts in hidden states caused by block removal. Hopscotch requires no changes to model weights and no access to pretraining or instruction-tuning data. Additionally, it is compatible with fine-grained compression techniques, such as model sparsification (Frantar and Alistarh, 2023) or KV cache quantization (Liu et al., 2024; Hooper et al., 2024; Wang et al., 2025), and can be combined with them to further reduce LLM inference costs.

We evaluate Hopscotch on instruction-tuned models, including `Llama-3.1-8B-Instruct` and `Qwen2.5-7B-Instruct`, and find that it can successfully skip up to 7 attention blocks with less than a 3% average performance drop across diverse tasks while yielding up to 15% inference speedup (see Section 4.6). With 4 blocks removed, we retain over 98.6% of baseline accuracy. These results highlight structural redundancy in attention blocks. We measure distributional shift in hidden representations and find that Hopscotch significantly reduces deviation from the original model, compared to unscaled attention block skipping. Finally, we show that Hopscotch is compatible with leading quantization techniques, such as GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2024).

2 Related Work

Feature scaling in generative models. Our work is inspired in part by recent advances in feature modulation techniques in generative models. For example, FreeU (Si et al., 2024) enhances image quality in diffusion U-Nets by scaling hidden states and skip connections and Ma et al. (2024) show that channel-wise scaling during post-training inference improves diffusion sampling quality. Our

[1]This name is inspired by the classic game where players hop through numbered squares, skipping the one with the marker.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 15904–15913

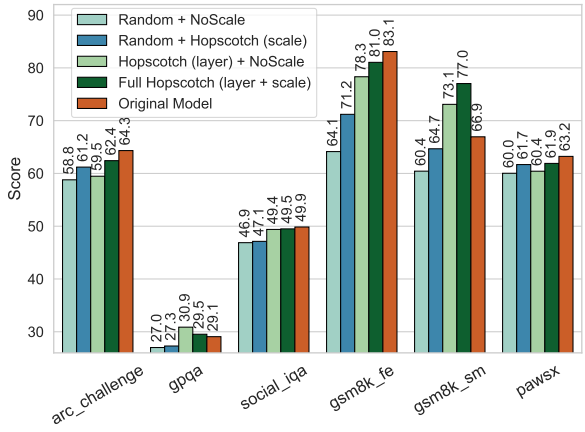

Figure 1: Comparison of benchmark performance when four attention blocks from layers (14, 17, 21, 24) are removed, before and after training scaling factors using Hopscotch. We also report results from Hopscotch's full pipeline: block selection and rescaling. Baseline scores from the original model without any blocks removed are included for reference.

work extends this direction by proposing dynamic rescaling of intermediate features in LLMs to compensate for pruned attention blocks.

Skipping computation in LLMs. Several recent studies have explored the possibility of reducing computation in LLMs by selectively skipping parts of the model. Shukor and Cord (2024) show that in multimodal LLMs, certain attention blocks contribute less to performance in vision-language tasks and can be skipped in these cases with minimal degradation. Further investigations on the true impact of attention blocks at different depths (Ben-Artzy and Schwartz, 2024; He et al., 2024) find that the effect is highly block-specific. Our work builds on these insights by providing a principled mechanism to identify and compensate for less informative attention blocks using learned scaling.

Model compression. Model compression techniques, including pruning, sparsification, and quantization, have been widely studied as means to reduce the memory and computational footprint of LLMs (e.g., Frantar and Alistarh, 2023; Sun et al., 2023; Frantar et al., 2022; Xiao et al., 2023; Yao et al., 2022; Ashkboos et al., 2024; Chee et al., 2023). These methods have shown impressive reductions in model size with minimal performance loss. Our approach is orthogonal to these methods, and can be used separately or in combination to further enhance model efficiency.

3 The Hopscotch method

Problem setup. When skipping one attention block, the input distribution to the next layer changes. To compensate the change, for each layer, we introduce four trainable scalar factors: one each for the outputs of the attention block, the MLP block, and the two residual connections. Let $\boldsymbol{x}_{\text{in}}^{(l)} \in \mathbb{R}^{n\times d}$ denote the input to the l-th transformer block, where n is the sequence length and d is the hidden dimension. Each transformer block is modified as follows:

$$\boldsymbol{x}_1^{(l)} = b_{\text{att}}^{(l)}\,\text{Attention}(\text{Norm}(\boldsymbol{x}_{\text{in}}^{(l)})) + s_{\text{att}}^{(l)}\boldsymbol{x}_{\text{in}}^{(l)}$$

$$\boldsymbol{x}_{\text{out}}^{(l)} = b_{\text{mlp}}^{(l)}\,\text{MLP}(\text{Norm}(\boldsymbol{x}_1^{(l)})) + s_{\text{mlp}}^{(l)}\boldsymbol{x}_1^{(l)}.$$

Setting all scaling factors to 1.0 recovers the original model, while setting any factor to 0.0 disables (i.e., remove) the corresponding component.

Loss function. Suppose we have an instruction-tuning dataset $\{x^{(i)}\}_{i=1}^{N}$, where each $x^{(i)}$ is a prompt consisting of an instruction and its input. We use the LLM on which we aim to skip attention blocks to generate the corresponding response $y^{(i)}$. Let L be the number of transformer blocks. We define the following loss function to learn the scaling factors $\theta = \left\{b_{\text{att}}^{(l)}, s_{\text{att}}^{(l)}, b_{\text{mlp}}^{(l)}, s_{\text{mlp}}^{(l)}\right\}_{l=1}^{L}$:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(-\sum_{t=1}^{T_i}\log P_\theta(y_t^{(i)} \mid y_{<t}^{(i)}, x^{(i)})\right).$$

The model weights are frozen, and only the scaling factors are updated during training. See Appendix A.1 for further details on loss function selection and recovery data.

3.1 Greedy Iterative Algorithm

At the heart of the Hopscotch method is a greedy iterative algorithm. In each iteration, we identify an attention block whose removal minimally affects the model's output, remove it, and then rescale the remaining blocks to compensate. This process is repeated until a target number of blocks are pruned or a performance threshold is reached. Below, we provide the detailed procedures.

Selecting attention blocks to remove. To select an attention block for removal, we estimate the impact of removing each block on the model's loss. Specifically, we define the impact of removing the l-th attention block as: $\min \mathcal{L}(\theta)$ s.t. $b_{\text{att}}^{(l)} = 0$. Solving this optimization exactly for *every* layer is computationally expensive. Instead, we approximate it by running a single optimization epoch with a large learning rate, quickly estimating which layer can be removed with minimal degradation.

Method		gsm8k FE	gsm8k SM	arc_challenge	gpqa	social_iqa	pawsx
Baseline		83.10	66.94	64.33	29.07	49.85	63.26
4 Blocks	Hopscotch	**81.05**	**77.03**	**62.42**	**29.53**	**49.49**	**61.90**
	ShortGPT	62.33	58.45	57.00	28.44	47.59	59.49
	FinerCut	75.80	61.00	53.58	27.77	46.93	54.98
7 Blocks	Hopscotch	**79.38**	**75.82**	**61.17**	**29.39**	**48.93**	**60.44**
	ShortGPT	1.90	1.29	46.42	27.35	43.76	57.62
	FinerCut	42.50	6.80	52.65	28.10	46.47	54.87
10 Blocks	Hopscotch	**67.93**	**62.58**	**57.80**	**29.83**	**48.23**	**61.06**

Table 1: Accuracy (%) of `Llama-3.1-8B-Instruct` on various benchmarks after skipping attention blocks (or full decoder blocks for ShortGPT). The table shows the scores of the original model compared to Hopscotch as well as prior methods when skipping 4, 7, or 10 blocks.

Rescaling the remaining blocks. Once an attention block is removed, we rescale the remaining blocks to recover model performance. This is done by minimizing the loss function $\mathcal{L}(\theta)$ over multiple epochs using a small learning rate. This longer optimization process adjusts the scaling factors to best compensate for the removed block and recovers model quality. Exact learning rates and hyperparameters can be found in Appendix A.2.

4 Numerical Experiments

4.1 Benchmarks and Setup

To assess the affects of attention block skipping and the Hopscotch method, we took a sample set of benchmarks: ARC Challenge (Abstraction and Reasoning) (Clark et al., 2018), GPQA (Challenging Google-Proof QA) (Rein et al., 2024), Social IQA (Commonsense Social Reasoning) (Sap et al., 2019), GSM8K (General Math) (Cobbe et al., 2021), PAWS-X (Multilingual) (Yang et al., 2019).

We run Hopscotch on `Llama-3.1-8B-Instruct` and `Qwen2.5-7B-Instruct` (see Appendix A.3 for details on model choice). As shown in Figure 2, during the attention block removal step, the removal of initial blocks results in relatively small, approximately linear increases in loss up to the seventh block for Llama and the fourth for Qwen. Based on this observed inflection in the loss curve, consistent with the elbow method heuristic (Wu et al., 2022), we use four and seven block removals as representative configurations for our evaluations.

Figure 1 compares the benchmark performance of our full method, Hopscotch layer selection with scaling, against four baselines: (i) the original unmodified model (baseline), (ii) random block removal with no scaling, (iii) random block removal with Hopscotch scaling, and (iv) Hopscotch-selected layers with no scaling. When simply removing sets of arbitrary attention blocks near the center of the model, performance remained within a few points for all but GSM8K, which degraded drastically. When using Hopscotch scaling, even without layer selection, we were able to recover a significant portion of the GSM8K performance, while also seeing slight improvement in other domains. For this reason, we use the math domain and GSM8K as our primary test case and training recovery set for the Hopscotch method, while continuing to evaluate on the full set of benchmarks.

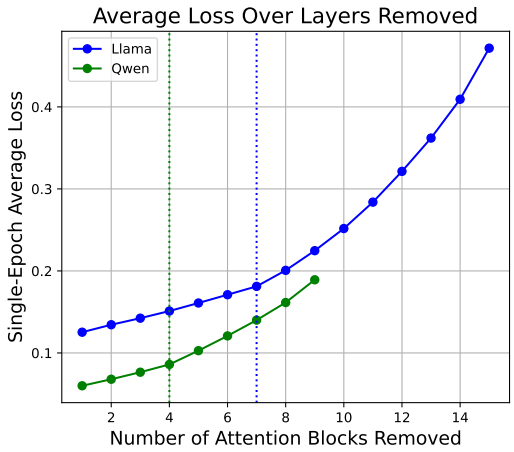

Figure 2: A plot of the single-epoch average loss per block removed during step one of Hopscotch for Llama.

4.2 Results on Llama-3.1-8B-Instruct

In Table 1, we present the results for `Llama-3.1-8B-Instruct` with four and seven blocks removed via Hopscotch, evaluated on the full benchmark suite. With seven blocks removed, we observe an average benchmark performance retainment of 97.08% when ignoring the GSM8K-SM (strict match) increase, and including it we see an average of 99.78%. With four blocks removed, average perfor-

mance in this case drops only 1.35% with a 98.65% overall recovery even when excluding strict match score. With strict match included average score actually goes up by 1.39%.

We also compare directly to two leading prior works: SparseGPT (Frantar and Alistarh, 2023) and FinerCut (Zhang et al., 2024). Hopscotch consistently outperforms both, especially on challenging tasks such as GSM8K, where other methods degrade significantly after removing four blocks and fail entirely beyond seven. Against ShortGPT, a key note is that the method removes entire layers, rather than keeping MLP blocks intact. This means for a fair comparison in terms of inference-time efficiency, one should compare ShortGPT with four layers removed to Hopscotch with at least six attention blocks removed (see Section 4.6 on how attention blocks contribute to inference time). We show that even with seven blocks removed, Hopscotch still outperforms across the board. For a closer comparison in terms of parameters removed, with 10 attention blocks removed, Hopscotch corresponds to about 83.3% of the parameter reduction by ShortGPT with 4 layers, yet achieves a relative inference speedup of $\mathbf{165.06}\%$. In this setting, Hopscotch still outperforms ShortGPT with 4 layers across all benchmarks (see Table 1). FinerCut also only removes attention blocks, making comparisons balanced with equal block removal. The primary remaining differences lie in the method of candidate block selection, as well as the critical discovery of the importance of scaling parameters.

4.3 Results on Qwen-2.5-7B-Instruct

Our findings also hold when using `Qwen2.5-7B-Instruct`. Table 2 shows the results for four blocks removed via the Hopscotch method. We observe an average recovery of 98.1% in benchmark performance when excluding GSM8K-SM (strict match). On-task GSM8K performance shows essentially perfect recovery. This is not surprising, as GSM8K training data was used to learn the scaling factors in Hopscotch, though the level of recovery notably exceeds the Llama experiments.

4.4 Measuring Distributional Shift

Removing attention blocks introduces a shift in hidden state representations throughout the model. To quantify this shift and evaluate how Hopscotch scaling effectively mitigates it, we compute the Maximum Mean Discrepancy (MMD) between hidden states in the original `LLaMA-3.1-8B-Instruct` model and its modified variants. MMD is a non-parametric metric used to quantify the difference between two distributions based on samples drawn from them. Specifically, we compare the MMD between (i) the original model and the version with zeroed attention blocks and no rescaling ("NoScale"), and (ii) the original model and the Hopscotch-scaled model.

Bench	4 Blocks	Baseline	Recovery
gsm8k FE	73.16	73.24	99.89%
gsm8k SM	39.20	17.74	220.97%
arc_chall	57.51	59.81	96.15%
gpqa	29.53	29.87	98.86%
social_iqa	43.81	45.96	95.32%
pawsx	60.14	59.97	100.28%

Table 2: `Qwen2.5-7B-Instruct` performance (accuracies) after skipping 4 attention blocks via Hopscotch.

Layer #	28	26	23	22	21	19	4
NoScale	0.62	0.50	0.40	0.38	0.33	0.26	0.35
Ours	**0.40**	**0.31**	**0.25**	**0.22**	**0.19**	**0.11**	**0.34**

Table 3: MMD scores across layers for two intervention methods. First row: Original vs. NoScale; Second row: Original vs. Ours.

As shown in Table 3, Hopscotch consistently yields lower MMD from the original model than NoScale across all layers, indicating reduced distributional shift. For instance, in layer 19, MMD drops from 0.2598 (NoScale) to 0.1098 with Hopscotch, a reduction of over 57%. These results support the core mechanism of Hopscotch: post-hoc scaling restores internal representations after attention block removal.

4.5 Compatibility with Quantization

To evaluate the compatibility of Hopscotch with other post-training model compression methods, we consider two state-of-the-art quantization techniques: GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2024). We test two strategies: (i) applying Hopscotch after quantizing the model, and (ii) applying Hopscotch before quantization. For both approaches, we report performance for 4 and 7 skipped attention blocks. As shown in Table 4, Hopscotch remains effective when combined with quantization irrespective of the application order. It consistently improves strict match accuracy and

achieves flexible extract performance comparable to the original uncompressed model.

Method	Strict Match	Flexible Extract
Baseline (Instruct)	67.0	83.9
GPTQ	68.0	83.9
GPTQ + Hopscotch (4)	77.3	79.4
GPTQ + Hopscotch (7)	75.1	78.5
Hopscotch (4) + GPTQ	77.9	79.2
Hopscotch (7) + GPTQ	73.8	78.5
AWQ	65.0	81.2
AWQ + Hopscotch (4)	71.4	75.1
AWQ + Hopscotch (7)	69.5	73.8
Hopscotch (4) + AWQ	75.2	78.3
Hopscotch (7) + AWQ	73.1	79.2

Table 4: GSM8K evaluation results for `LLaMA-3.1-8B-Instruct` using Hopscotch in combination with GPTQ and AWQ quantization.

4.6 Effects on Model Efficiency

In a given forward pass for `Llama-3.1-8B-Instruct`, we experimentally find that $\sim$66% of time is spent in decoder attention blocks. In a model with 32 hidden layers, this means $\sim$2.06% of the forward pass is spent in each block. For example, for a query with an average forward pass time of 0.054 seconds, the time spent in a single attention block is on average 0.0011 seconds. The effect of removing a subset of attention blocks on overall inference time is summarized in Table 5.

Attn. Blocks Removed	Inference Time Reduction
1	2.06%
4	8.24%
7	14.42%

Table 5: Inference-time performance gains compared to blocks removed for `Llama-3.1-8B-Instruct`.

It is also worth noting that when removing these attention blocks, we no longer have to store the corresponding parameters in memory, resulting in reduced-size models with lower GPU memory footprints. Each decoder layer in `Llama-3.1-8B-Instruct` consists of an attention block and an MLP block. The attention block includes four linear projections (query, key, value, and output), each with weight matrices of shape $h \times h$ (neglecting biases), contributing $4h^2$ parameters per layer. The MLP block typically includes two projections: $h \times 4h$ and $4h \times h$, totaling $8h^2$ parameters. Therefore, the attention block accounts for $\frac{4h^2}{4h^2+8h^2} = \frac{1}{3} \approx 33.3\%$ of the parameters in a decoder layer.

Given that `Llama-3.1-8B-Instruct` has 32 decoder layers and a total of $\sim$8B parameters, each layer contains approximately 250M parameters. Removing one attention block saves roughly 83M parameters, which is about 1.04% of the model. Removing seven attention blocks yields a total reduction of $7 \times 83\text{M} = 581\text{M}$ parameters, or about 7.28% of the total model size. Since each parameter occupies 2 bytes in float16, we can estimate the memory savings accordingly (Table 6).

Attn. Blocks Removed	Parameters Reduction	Memory Reduction
1	1.04%	$\sim$0.17 GB
4	4.16%	$\sim$0.67 GB
7	7.28%	$\sim$1.16 GB

Table 6: Estimated memory savings by removing attention blocks from `Llama-3.1-8B-Instruct` (assuming 2 bytes per parameter in float16 thus total parameter memory of 16 GB).

Further memory savings will be observed in practice due to reductions in optimizer states during training (e.g., Adam requires 8–12 bytes per parameter, leading to approximately 3$\times$ the model size), activation memory for forward and backward passes, gradient storage which is typically the same size as the model, and the KV cache used during inference with long contexts, which can consume 2–4 GB per 1,000 tokens for large models. This also means batch sizes can be potentially increased, further boosting inference gains.

5 Conclusion

We introduce Hopscotch, a simple yet effective method for skipping attention blocks in LLMs. To preserve model outputs, Hopscotch learns scaling factors for attention and MLP blocks in remaining layers, compensating for distributional changes in hidden states. We present promising results, achieving near-lossless performance on standard benchmarks. Hopscotch is also compatible with existing model compression techniques. We hope this work paves the way for future research on identifying and reducing redundant computations in attention mechanisms to build more efficient foundation models.

6 Limitations

We focus exclusively on the attention mechanism, including multi-head, multi-query, and group-query variants, which are widely adopted in open-source LLMs such as Llama and Qwen. However, emerging architectures like multi-head latent attention and state space models such as Mamba (Gu and Dao, 2023) offer new directions. It would be interesting to explore whether similar observations (e.g., skipping certain components) hold for these architectures as well. Additionally, there is an inherent trade-off between model compression and performance. When deploying compressed models in high-stakes applications that require high precision (such as disease detection) it is important to rigorously evaluate their performance to prevent potential harm.

7 Ethical Considerations

LLMs are growing which raises concerns about the accessibility and inclusiveness of AI development. Large models impose significant computational and memory demands, as well as increased infrastructure requirements. As a result, innovation increasingly concentrates within a few well-funded organizations, limiting participation from individual developers, smaller labs, and open-source community. This not only restricts opportunities for broader collaboration but also challenges the ability of open-source users to benefit from and contribute to cutting-edge model development. In this paper, we aim to bridge this gap by mitigating the inference cost of LLMs through architectural modifications, specifically, by identifying and skipping redundant attention blocks. Our hope is to contribute to a more equitable landscape in AI development, one where more individuals and institutions can meaningfully participate in and benefit from state-of-the-art LLMs.

References

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240.

Amit Ben-Artzy and Roy Schwartz. 2024. Attend first, consolidate later: On the importance of attention in different llm layers. *arXiv preprint arXiv:2409.03621*.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2023. Quip: 2-bit quantization of large language models with guarantees. *Advances in Neural Information Processing Systems*, 36:4396–4429.

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. *arXiv preprint arXiv:1803.05457*.

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

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

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*.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. The language model evaluation harness.

IBM Granite Team. 2024. Granite 3.0 language models.

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*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. 2024. What matters in transformers? not all attention is needed. *Preprint*, arXiv:2406.15786.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. 2020. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. 2017. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. 2024. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization. *Preprint*, arXiv:1412.6980.

Achintya Kundu, Rhui Dih Lee, Laura Wynter, Raghu Kiran Ganti, and Mayank Mishra. 2024. Enhancing training efficiency using packing with flash attention. *Preprint*, arXiv:2407.09105.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32332–32344.

Jiajun Ma, Shuchen Xue, Tianyang Hu, Wenjia Wang, Zhaoqiang Liu, Zhenguo Li, Zhi-Ming Ma, and Kenji Kawaguchi. 2024. The surprising effectiveness of skip-tuning in diffusion sampling. *arXiv preprint arXiv:2402.15170*.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Mustafa Shukor and Matthieu Cord. 2024. Skipping computations in multimodal llms. *arXiv preprint arXiv:2410.09454*.

Chenyang Si, Ziqi Huang, Yuming Jiang, and Ziwei Liu. 2024. Freeu: Free lunch in diffusion u-net. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4733–4743.

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*.

Hao Wang, Ligong Han, Kai Xu, and Akash Srivastava. 2025. Squat: Subspace-orthogonal kv cache quantization. *arXiv preprint arXiv:2503.24358*.

Zhihong Wu, Fuxiang Li, Yuan Zhu, Ke Lu, Mingzhi Wu, and Changze Zhang. 2022. A filter pruning method of cnn models based on feature maps clustering. *Applied Sciences*, 12(9):4541.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. 2019. Paws-x: A cross-lingual adversarial dataset for paraphrase identification. *arXiv preprint arXiv:1908.11828*.

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*, 35:27168–27183.

Yang Zhang, Yawei Li, Xinpeng Wang, Qianli Shen, Barbara Plank, Bernd Bischl, Mina Rezaei, and Kenji Kawaguchi. 2024. Finercut: Finer-grained interpretable layer pruning for large language models. *Preprint*, arXiv:2405.18218.

A Appendix

A.1 Data and Loss Function Selection

Using GSM8K training data, we evaluated two different options for the ground-truth answers to be used in training. The first was to use the original ground-truth answers provided with the dataset. The second was to instead use the baseline model's greedy generations as the ground-truth answers, since the goal is to recover original model performance. To evaluate between the two options, we train with all scaling factors left free (no blocks removed) on both datasets. As we can see in Table 7, there seems to be a significant performance difference between the two options, with the model-generations-as-ground-truth scoring higher in both categories (flexible extract and strict match) for GSM8K. Additionally, we actually see improvement in strict match performance over the *baseline* when doing model-generations-as-ground-truth training. We will see later as well that this holds even when blocks are removed from the model.

Method	Flexible Extract	Strict Match
Baseline	**83.10**	66.94
Model GT	82.79	**77.48**
Data GT	60.88	42.61

Table 7: Comparison of different methods based on Flexible Extract and Strict Match metrics.

This behavior makes sense given the goal of the training. When using the original data ground truth, we risk introducing *new information* not previously in the model's distribution, and trying to learn that new information within a set of 128 total parameters could (and in this case did) lead to significant model degradation. Instead, what we are trying to accomplish is simply learning how to accent and distribute existing information to match expected steps, reasoning, formatting, etc. We are building a map from the existing attention blocks to the original known distribution, by learning from in-distribution outputs to find the appropriate attention block weighting. We are appropriately reweighting the blocks in our model to approximate that original distribution, and so we need to ensure that we are learning exactly and only that reverse-mapping from output to block focus, rather than introducing anything the model was not originally capable of producing.

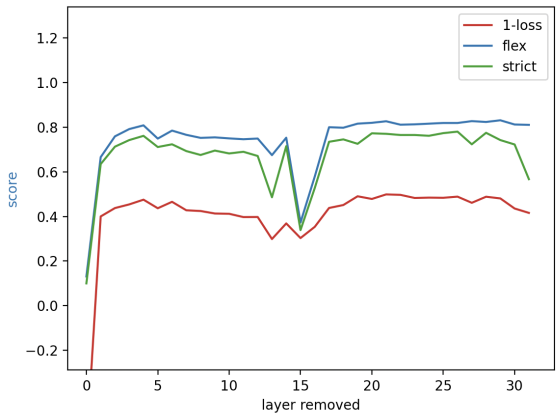

Figure 3: Correlation between average model-ground-truth cross-entropy loss given a layer with attention block removed and benchmark performance on GSM8K (flexible extract and strict match).

With the training data established, the next major piece to determine in the training dynamics was the actual loss function utilized. There were two options considered. The first was to use the standard Cross-Entropy loss typically employed in Causal LM training, in conjunction with our generated model-ground-truth answers. The second was to take things a step further, and instead use a loss calculated by the squared L2 norm between the final *hidden states* (output of the final hidden layer) of the original model for a given input against the final hidden states of the model with removed attention blocks. This second method proved to be too rigid, however, as it's optimality relied on the assumption that the only path to a similarly correct final output was via a similar final hidden state.

Overall, the combination of model-outputs-as-ground-truth and Cross-Entropy loss resulted in promising initial results and high correlation to benchmark performance, with the training loss providing a Spearmann correlation of 0.9 for the GSM8K benchmark scores (Figure 3).

A.2 Hyperparameters and Training Details

Training is done using padding-free sample packing (Kundu et al., 2024) with a batch size of 32, Adam optimizer (Kingma and Ba, 2017), and a learning rate of 3e-3 for scaling parameter training, and 1e-2 for attention block selection. GSM8K training data consisted of 7473 samples. Evaluation done via LM Eval Harness (Gao et al., 2024).

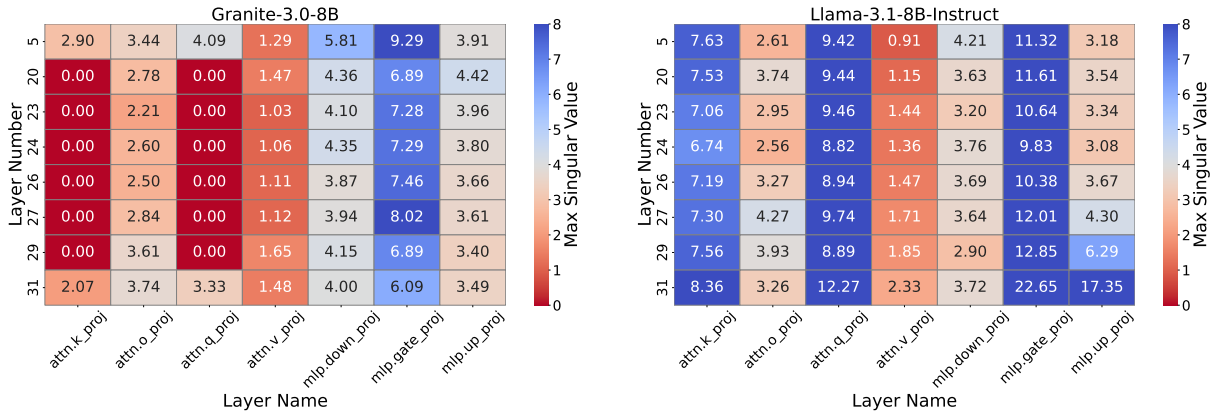

Figure 4: We show the largest singular values of weight matrices for a representative subset of layers to highlight structural differences between models. These values serve as a proxy for quantifying each block's contribution to the model's output. We present results using the `Granite-3.0-8B` model (left) and the `Llama-3.1-8B-instruct` model (right). Red indicates low singular values (near 0), while blue indicates high values (8 and above).

A.3 Base Model Selection

It is important to note that the selection of `Llama-3.1-8B-Instruct` as a base model was intentional, as it serves as an example of a model where all layers meaningfully transform the hidden states and contribute to the model's outputs, better showing the generalizability of Hopscotch. For example, we show that with a model like our fine-tuned `Granite-3.0-8B`, the process of selecting attention blocks to remove is trivial, as there are a number of non-contributing blocks in the model. We illustrate why it is possible to skip attention blocks by analyzing the largest singular values of the weight matrices across different layers. Singular values quantify how much a transformation can stretch or distort the input. Specifically, for a linear transformation $\boldsymbol{x} \rightarrow \mathbf{W}\boldsymbol{x}$, the inequality $\|\mathbf{W}\boldsymbol{x}\|_2 \leq \sigma_{\max}(\mathbf{W})\|\boldsymbol{x}\|_2$ implies that if $\sigma_{\max}(\mathbf{W})$ is small and $\boldsymbol{x}$ is bounded, the output of the transformation will be near zero. This suggests that skipping such transformations would have minimal effect on the model's output.

We present results using two open-source LLMs: `Granite-3.0-8B` (Granite Team, 2024) and `Llama-3.1-8B-Instruct` (Grattafiori et al., 2024) in Figure 4. The Granite model contains layers with negligible singular values, suggesting limited contribution to the output. `Llama-3.1-8B-Instruct` generally has larger and more consistent singular values. When we remove blocks from Llama, we see a non-negligible drop in benchmark performance. With Granite, however, we see in Table 8 that removing attention blocks from layers with maximum singular values of zero results in no loss in model performance for the case of GSM8K.

Bench	6 Blocks Skipped	Baseline
gsm8k (flex.)	75.66	74.00
gsm8k (strict)	66.11	64.90

Table 8: Comparing benchmark performance when six attention blocks from layers (20, 23, 24, 26, 27, 29) are skipped, without any additional re-scaling. Includes original baseline scores before attention blocks were removed.

A.4 Using Loss as an Approximation of Benchmark Performance

As seen in Figure 3, training loss when removing blocks serves as strong indication of how the removal will affect overall model performance. Thus, at any given model state, we can know the best attention block to remove by finding the block that, when removed, provides the lowest loss. Since we simply need an indication via quick conversion, rather than an optimized loss, we can simply run our training method for a single epoch with an increased learning rate (1e-2), and take the average training loss as a comparable value for each block.

A.5 Demonstrating Iterative Greedy Value

As can be seen in Table 9, when using the greedy iterative approach, the attention blocks removed prove to retain significantly more performance than taking a full greedy approach and assuming layer independence. With nine attention blocks removed from each, the average loss is notably lower for the

Method	Average Loss (9 Blocks)
Full Greedy	.2765
Iterative Greedy	**.2247**

Table 9: Comparison of single-epoch average loss after nine attention blocks removed via full greedy approach and iterative greedy approach.

greedy iterative approach. The iterative approach could remove two more attention blocks before reaching a similar loss to the full greedy approach.