EVOPRESS: ACCURATE DYNAMIC MODEL COMPRESSION VIA EVOLUTIONARY SEARCH

Oliver Sieberling
ETH Zürich
osieberling@ethz.ch

Denis Kuznedelev
Yandex & Skoltech
denis.kuznedelev@skoltech.ru

Eldar Kurtic
IST Austria & Neural Magic
eldar.kurtic@ist.ac.at

Dan Alistarh
IST Austria & Neural Magic
dan.alistarh@ist.ac.at

ABSTRACT

The high computational costs of large language models (LLMs) have led to a flurry of research on LLM compression, via methods such as quantization, sparsification, or structured pruning. A new frontier in this area is given by *dynamic, non-uniform* compression methods, which adjust the compression levels (e.g., sparsity) per-block or even per-layer in order to minimize accuracy loss, while guaranteeing a global compression threshold. Yet, current methods rely on estimating the "importance" of a given layer, implicitly assuming that layers contribute independently to the overall compression error. We begin from the motivating observation that this independence assumption does not generally hold for LLM compression: pruning a model further may even significantly recover performance. To address this, we propose EvoPress, a novel evolutionary framework for dynamic LLM compression. By formulating compression as a general optimization problem, EvoPress identifies optimal compression profiles in a highly efficient manner, and generalizes across diverse models and compression techniques. Via EvoPress, we achieve state-of-the-art performance for structured and unstructured compression of Llama, Mistral, and Phi models.

1 INTRODUCTION

Compression via elimination of the least important parameters has become a standard way of reducing the deployment costs of large language models (LLMs). Pruning techniques can be categorized into unstructured, which sparsify the weight matrices, e.g. Frantar & Alistarh (2023); Yin et al. (2024), or structured pruning / layer dropping, which drop entire model components, e.g. Kim et al. (2024); Men et al. (2024). While improvements are still being made, existing methods are reaching diminishing returns in terms of accuracy-vs-compression (Dettmers et al., 2023; Tseng et al., 2024).

Prior approaches for unstructured sparsity (Yin et al., 2024; Frantar & Alistarh, 2022) and layer dropping (Kim et al., 2024; Men et al., 2024) work by assigning an *error/sensitivity score* to each layer, which measures the impact of its compression on the output loss increase. Then, one calculates a compression assignment which minimizes the sum of error scores, while still satisfying the global compression constraint. Thus, such approaches inherently assume *error monotonicity*: i.e., that a lower sum of error scores taken over layers implies a lower compression error for the entire model.

Our work starts from the observation that error monotonicity *does not hold* generally for LLM compression: specifically, there are instances where *compressed models with lower sums of per-layer errors can perform* worse *than models with higher error*. We illustrate this fact in Table 1, which shows an instance of a layer dropping configuration where pruning *more blocks* leads to significantly better perplexity than an instance which prunes *strictly less* blocks.

Related Work. Recently, there has been a lot of interest in compression by removing entire transformer blocks, both for efficiency and to gain insights about the language model itself. Most methods are based on scoring the importance of each block, and then maximizing the importance of the resulting model by removing the blocks of lowest importance. Therefore, they assume error

Table 1: Depth pruning is not monotone. In this example (Llama-3-8B), removing strictly more blocks (depicted in orange) *can improve* perplexity across sources. The left half of a block corresponds to the attention layer, the right half to the MLP.

Model	Configuration (Each block contains Attention + MLP)	Wiki2↓	C4↓	FW↓
Llama-3-8B		5.54	8.80	7.72
		188.01	147.25	70.46
		**24.39**	**35.53**	**26.24**

linearity, meaning that each block contributes independently to the total compression error. Weight Subcloning (Samragh et al., 2023) proposed a multi-step process to find good initializations for an untrained smaller model given an already trained larger one, where the importance of each block is scored based on the ratio of ℓ_2 norms between the output embeddings of the block with and without the residual connection. Shortened Llama (Kim et al., 2024) proposes scoring each block by measuring the perplexity after removing the respective block from the full model. ShortGPT (Men et al., 2024) uses the cosine similarity between the input and output embeddings of each block to assess its importance. By contrast, Gromov et al. (2024) restrict to removing *consecutive blocks* and score each configuration by cosine similarity.

Contribution. To address this, we propose a new evolutionary search approach called EvoPress, which identifies effective compression profiles while being sample and iteration efficient. These two efficiency properties are critical for practicality in the context of LLMs, where the cost of evaluating single models ("offspring") is exceedingly high. We validate the approach for layer dropping and one-shot sparsification. We find that EvoPress consistently improves upon existing techniques, with major improvements at higher compression ratios. Moreover, it can do so efficiently: when applied to layer dropping for an 8B-parameter model, EvoPress converges in a matter of minutes on a single RTX 3090 GPU.

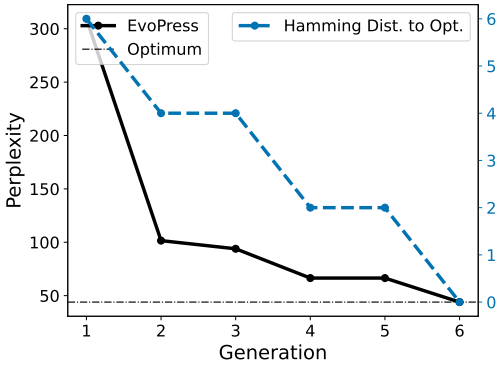

Figure 1: Removing 12 transformer blocks from Llama-3-8B under the constraint that only pairs of consecutive blocks can be removed. EvoPress finds the optimal configuration from the 8008 possible removal combinations in generation 6.

2 METHOD

All applications of EvoPress are grounded in a unified framework, where the objective is to identify the optimal model that adheres to a specified compression method and constraint. Formally, given a base model M, we seek to maximize the performance of the compressed model while satisfying the compression constraint:

$$\hat{M}^* = \arg\max_{\hat{M}} f(\hat{M}) \quad \text{subject to} \quad g(\hat{M}) \leq C,$$

where $f(\hat{M})$ quantifies the performance of the compressed model $\hat{M}$ and $g(\hat{M})$ represents the compression constraint. For simplicity, we will define g as the model's total size (in terms of parameters); however, the method can be adapted to other constraints, such as inference speed.

We approach this optimization problem using evolutionary search, which is a specific form of randomized search. The feasibility of such an approach heavily depends on two factors: the time required to evaluate the fitness of a candidate solution and the number of such function evaluations needed until a satisfying result is achieved. This poses a particular challenge in our case, as assessing the performance of an LLM involves substantial computational costs.

Fitness Environment. Given the specified database, any compressed model is characterized by its compression level per unit (e.g. per layer). With n units, each available in m compression levels, our objective is to find

$$\hat{M}^* = \arg\max_{v \in [m]^n} f(\hat{M}_v) \quad \text{subject to} \quad g(\hat{M}_v) \leq C,$$

where we are searching over the set of n-tuples over $[m]$. Assessing the performance of a model in practice typically involves benchmark tasks, which have limited scope and require lengthy evaluation. We address these challenges by using the base model as the gold standard and focusing solely on the relative degradation of our compressed models. To quantify this degradation, we measure the Kullback-Leibler (KL) divergence between the two models, as it has proven particularly robust with limited data. Empirically, we observed that already around 64K tokens of calibration data (corresponding to 8 full sample sequences for Llama-3-8B) are sufficient to reliably determine the quality of the lightweight model. To avoid confusion, we will refrain from inverting the fitness function and from now on consider the minimization problem

$$\hat{M}^* = \arg\min_{v \in [m]^n} D_{KL}(P_M \parallel Q_{\hat{M}_v}) \text{ subject to } g(\hat{M}_v) \leq C,$$

where we speak of *higher fitness* whenever the KL-Divergence is *lower*.

Algorithm. EvoPress starts from upon the classic $(1 + \lambda)$-evolutionary algorithm, which maintains a single search point at any given time. In each generation, λ offspring are generated by copying the parent and then applying a mutation operator to each copy. The offspring are then evaluated on the fitness function, and the fittest one is selected. As an *elitist* evolutionary algorithm, the $(1 + \lambda)$-EA replaces its parent only if the best offspring has superior fitness.

We change this standard algorithm in two important ways. The first is by introducing *level-switch mutation*, a simple mutation operator that ensures high locality while preserving the compression constraint. The operator involves first randomly selecting one unit and increasing its compression level. Next, a second unit is sampled until one with a matching level step size is found, and its compression level is decreased. This approach ensures that 1) the compression constraint is preserved, and 2) the offspring model maintains high similarity to the parent model – an important feature for achieving rapid convergence.

The second modification is that we employ a very aggressive form of *multi-step selection*. In the first stage, all λ offspring are evaluated using only a fraction of a full sample. From this, only a small subset of the fittest offspring are selected to compete in the next stage, where they are evaluated on a significantly larger sample size. This process is repeated once more, and in the final stage, the few remaining offspring are evaluated against the parent using a "full" minibatch, consisting of approximately 20-50 times the number of tokens used in the first stage. A high level overview of this procedure is presented in Appendix A.

3 EXPERIMENTS

We now validate the effectiveness of EvoPress for **layer dropping**, where the goal is to isolate the "optimal" set of blocks to drop given a target ratio. Results on unstructured sparsity are presented in Appendix C. We focus on LLM compression, given the major interest in the reduction of their model size and inference latency, but our method is general and can be applied to any neural network architecture and application domain.

Experimental Setup.

We consider base models from the Llama-2 and Llama-3 (Touvron et al., 2023) families, Mistral-v0.3 (Jiang et al., 2023), and the instruction-tuned Phi3-Medium-instruct-128k model (Abdin et al., 2024). We adopt KL-divergence as our fitness function because it provides a stronger and more robust signal compared to perplexity, better reflecting the predictive distribution of the original model. As a source of clean and diverse calibration data, which is required to evaluate fitness, we use Fineweb-Edu (Penedo et al., 2024).

Evaluation. We follow a standard evaluation protocol (Frantar et al., 2022), measuring perplexity on the WikiText-2 (Merity et al., 2016) and C4 (Raffel et al., 2019) datasets for language performance and accuracy on zero-shot evaluations on standard benchmarks: WinoGrande (Sakaguchi et al., 2021), PiQA (Tata & Patel, 2003), HellaSwag (Zellers et al., 2019), ARC-easy and ARC-challenge (Clark et al., 2018) via the LM Eval Harness (Gao et al., 2021).

Although removing entire transformer blocks generally results in large accuracy losses, this approach recently attracted attention in the context of initializing smaller models, as it guarantees speedups proportional to the sparsity (Samragh et al., 2023; Kim et al., 2024). Additionally, block dropping provides insights into the capabilities of transformer models, making it relevant for interpretability. We will compare against the following baselines: (1) **Shortened Llama** (Kim et al., 2024), which

scores blocks on the perplexity change after removal; (2) **ShortGPT** (Men et al., 2024), where blocks are scored based on the average cosine similarity between input and output embeddings, including the residual stream; (3) **Weight Subcloning** (Samragh et al., 2023), where blocks are scored using the ratio $||f(x)||/||f(x)+x||$, where x is the input embedding and $f(x)$ is the block's output, excluding the residual stream; (4) **Sliding Window Cosine Similarity** (Gromov et al., 2024), where sets of consecutive blocks are scored based on the cosine similarity between embeddings before and after the blocks, including the residual stream. While Gromov et al. (2024) directly score entire removal configurations, the other approaches determine block removals based on their isolated scores.

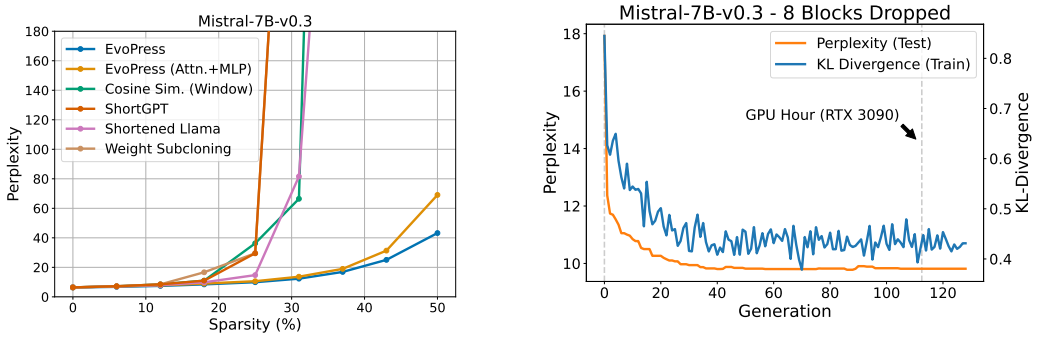

Figure 2: Depth pruning results, on Mistral-7B-v0.3. (**Left**) Relative to all prior methods, EvoPress shows significantly lower PPL gap relative to the uncompressed model, with remarkably large gaps at medium compression rates. (**Right**) Convergence of EvoPress when removing 8 transformer blocks.

Search Space. In our approach, attention and MLP modules are treated independently rather than as a single unit. For each module, there are two options: either retain it or remove it. To achieve a target sparsity/depth, we initially remove an equal number of attention and MLP modules. During mutation, we allow compression level adjustments only between modules of the same type. We leave it open for future research to remove this constraint to allow flexibility in the number of removed attention and MLP modules.

Results. Figure 2 compares our method with baselines from previous work on Mistral-7B-v0.3. For a better comparison, we also included results where only entire transformer blocks are removed (Attn.+MLP). EvoPress consistently outperforms all previous methods, showing significant improvements even at medium sparsity levels. While all baseline methods fail entirely beyond 31.25% sparsity, EvoPress identifies functional submodels even when removing half of the model. To our knowledge, this is the first method to achieve such results. We observed similar collapses in Llama-2-7B, Llama-3-8B and Llama-3.1-8B. Overall, EvoPress consistently outperforms all baselines across all tested models and sparsities (see Appendix B for full results), and does so within minutes.

All four previous methods rely on human-crafted scoring methods to identify the optimal combination of transformer blocks to remove. This is not only suboptimal, but also prone to bias, as their results may reflect the characteristics of the method itself rather than the model's true behavior. Specifically, we found that most scoring methods tend to favor deeper blocks, resulting in highly similar removal configurations across different prior scoring methods (see Table 9 in Appendix). This likely occurs because methods that bias towards deeper blocks generally perform better than those that focus on earlier blocks, although neither may be optimal. In contrast, EvoPress employs an unbiased approach, which offers more accurate and meaningful insights into the model.

4 Conclusion

We have presented EvoPress, an optimization framework for dynamic model compression. EvoPress is based on a new evolutionary search algorithm with low sample and iteration complexity, especially well-suited to loss landscapes in LLM compression. Specifically, we have shown that EvoPress can converge extremely fast to accurate configurations for structured and unstructured sparsity, and is also fast to execute in practice. Interesting directions we did not investigate are 1) combining *different compression approaches* into the same search space, and 2) finer-grained structured pruning. We plan to investigate this in future work.

REFERENCES

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Dan Iter, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xia Song, Masahiro Tanaka, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonali Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

Elias Frantar and Dan Alistarh. SPDY: Accurate pruning with speedup guarantees. *arXiv preprint arXiv:2201.13096*, 2022.

Elias Frantar and Dan Alistarh. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

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

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, September 2021. URL `https://doi.org/10.5281/zenodo.5371628`.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, and Hyoung-Kyu Song. Shortened llama: A simple depth pruning for large language models, 2024.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL `https://arxiv.org/abs/2406.17557`.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, 2021. doi: 10.1145/3474381. URL https://doi.org/10.1145/3474381.

Mohammad Samragh, Mehrdad Farajtabar, Sachin Mehta, Raviteja Vemulapalli, Fartash Faghri, Devang Naik, Oncel Tuzel, and Mohammad Rastegari. Weight subcloning: direct initialization of transformers using larger pretrained ones, 2023.

Sandeep Tata and Jignesh M Patel. PiQA: An algebra for querying protein data sets. In *International Conference on Scientific and Statistical Database Management*, 2003.

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

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks, 2024.

Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pechenizkiy, Yi Liang, Michael Bendersky, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1472. URL https://doi.org/10.18653/v1/p19-1472.

A ALGORITHM

Algorithm 1: EvoPress: A $(1+\lambda)$-Evolutionary Algorithm with Level-Switch Mutation and Multi-Step Selection for Maximizing a Fitness Environment $f : [m]^n \to \mathbb{R}$.

Initialization: `candidates` $\leftarrow []$;
for $i \leftarrow 1$ **to** *initialCandidates* **do**
  `candidate` $\leftarrow$ `sampleUniformly()`;
  `candidates.append(candidate)`;
$x^{(1)} \leftarrow$ `selectTopKFittest(candidates, initialTokens,` $K = 1$`)`;
Optimization: for $t \leftarrow 1$ **to** ∞ **do**
  `offspring` $\leftarrow []$;
  Mutation: for $i \leftarrow 1$ **to** λ **do**
    $y_i \leftarrow x^{(t)}$;
    $y_i \leftarrow$ `LevelSwitchMutation(`y_i`)`;
    `offspring.append(`y_i`)`;
  Selection: for *step* $\leftarrow 1$ **to** *selectSteps* **do**
    Elitism: if *step* $=$ *selectSteps* **then**
      `offspring.append(`$x^{(t)}$`)`;
    `offs.` $\leftarrow$ `selectTopKFittest(offs.,tokens[step],` $K =$ `survivors[step])`;
  $x^{(t+1)} \leftarrow$ `offspring`$[0]$;

B Additional Depth Pruning Results

Here, we present our additional results for depth pruning experiments on Mistral-7B-v0.3 (Table 5), Llama-2-7B (Table 2), Llama-3-8B (Table 3), and Llama-3.1-8B (Table 4). Across all levels of sparsities, EvoPress consistently outperforms previous methods. Additionally, Table 5 includes results where only entire transformer blocks are removed by EvoPress. This showcases that the significant gains are not primarily due to this relaxation, and that our method performs better than baselines even when dealing with this coarser search space.

Table 2: Depth pruning of Llama-2-7B.

Sparsity	Method	Wiki2↓	C4↓	FW↓
0%	Dense	5.21	6.93	6.40
12.5%	EvoPress	**6.42**	**8.60**	**7.54**
	ShortGPT	8.86	10.78	9.30
	Cosine Similarity (Window)	7.53	9.82	8.51
	Weight Subcloning	9.09	11.06	9.60
	ShortenedLlama	7.68	10.44	8.57
25%	EvoPress	**9.15**	**11.46**	**9.69**
	ShortGPT	23.41	30.30	21.16
	Cosine Similarity (Window)	16.60	21.04	17.37
	Weight Subcloning	23.41	30.30	21.16
	Shortened Llama	13.86	14.08	11.81
37.5%	EvoPress	**17.98**	**18.91**	**15.53**
	ShortGPT	70.94	63.51	54.07
	Cosine Similarity (Window)	192.07	212.60	151.10
	Weight Subcloning	70.94	63.51	54.07
	Shortened Llama	35.37	26.07	20.37
50%	EvoPress	**48.84**	**42.29**	**33.57**
	ShortGPT	226.14	171.04	180.51
	Cosine Similarity (Window)	4570.15	2876.83	1861.06
	Weight Subcloning	226.14	171.04	180.51
	Shortened Llama	145.78	87.40	68.79

Table 3: Depth pruning of Llama-3-8B.

Sparsity	Method	Wiki2↓	C4↓	FW↓
0%	Dense	5.54	8.80	7.62
12.5%	EvoPress	**7.72**	**12.61**	**10.15**
	ShortGPT	13.21	19.56	14.25
	Cosine Similarity (Window)	9.54	14.87	11.64
	Weight Subcloning	13.21	19.56	14.25
	Shortened Llama	9.42	15.09	11.57
25%	EvoPress	**13.99**	22.83	**15.84**
	ShortGPT	5527.54	11589.93	2346.13
	Cosine Similarity (Window)	5519.95	11629.61	2342.91
	Weight Subcloning	5527.54	11589.93	2346.13
	Shortened Llama	16.59	**20.81**	16.28
37.5%	EvoPress	**27.56**	**35.70**	**26.77**
	ShortGPT	64281.36	13836.12	3789.09
	Cosine Similarity (Window)	64627.29	13890.14	3784.72
	Weight Subcloning	64381.36	13836.13	3789.09
	Shortened Llama	50.20	61.56	37.40
50%	EvoPress	**84.99**	**87.86**	**66.41**
	ShortGPT	1663.97	1740.04	1588.20
	Cosine Similarity (Window)	2053.19	1116.47	694.00
	Weight Subcloning	1663.97	1740.04	1588.20
	Shortened Llama	724.86	666.41	210.30

Table 4: Depth pruning of Llama-3.1-8B.

Sparsity	Method	Wiki2↓	C4↓	FW↓
0%	Dense	5.61	8.90	7.67
12.5%	EvoPress	**7.58**	**12.24**	**10.00**
	ShortGPT	12.54	19.21	13.76
	Cosine Similarity (Window)	12.54	19.21	13.76
	Weight Subcloning	12.54	19.21	13.76
	Shortened Llama	9.27	14.80	11.21
25%	EvoPress	**11.59**	**17.84**	**13.96**
	ShortGPT	4278.39	6754.92	1512.39
	Cosine Similarity (Window)	4278.39	6754.92	1512.39
	Weight Subcloning	4278.39	6754.92	1512.39
	Shortened Llama	20.41	20.33	16.12
37.5%	EvoPress	**24.98**	**35.77**	**25.93**
	ShortGPT	123044.19	22071.51	6059.03
	Cosine Similarity (Window)	123044.19	22071.51	6059.03
	Weight Subcloning	123044.19	22071.51	6059.03
	Shortened Llama	41.34	43.53	31.00
50%	EvoPress	**105.84**	**110.69**	**61.25**
	ShortGPT	1630.11	1680.21	1698.64
	Cosine Similarity (Window)	1881.54	1196.63	683.24
	Weight Subcloning	1630.11	1680.21	1698.64
	Shortened Llama	454.96	309.42	153.96

Table 5: Depth pruning of Mistral-7B-v0.3.

Sparsity	Method	Wiki2↓	C4↓	FW↓
0%	Dense	4.82	7.72	6.41
12.5%	EvoPress	**6.06**	**9.00**	**7.42**
	EvoPress (Attn.+MLP)	6.33	9.44	7.80
	ShortGPT	7.19	10.18	8.46
	Cosine Similarity (Window)	7.19	10.18	8.46
	Weight Subcloning	7.19	10.18	8.46
	Shortened Llama	6.64	9.71	7.94
25%	EvoPress	**8.66**	**12.04**	**9.92**
	EvoPress (Attn.+MLP)	9.46	13.02	10.59
	ShortGPT	43.26	40.16	29.54
	Cosine Similarity (Window)	33.75	54.07	36.26
	Weight Subcloning	43.26	40.16	29.54
	Shortened Llama	14.94	19.30	14.73
37.5%	EvoPress	**17.52**	**21.60**	**16.90**
	EvoPress (Attn.+MLP)	21.62	25.17	18.97
	ShortGPT	2898.98	2722.66	981.99
	Cosine Similarity (Window)	1034.09	2471.86	1050.56
	Weight Subcloning	2898.98	2722.66	981.99
	Shortened Llama	440.20	442.09	486.15
50%	EvoPress	**61.75**	**54.15**	**43.23**
	EvoPress (Attn.+MLP)	108.91	99.74	69.07
	ShortGPT	2422.72	2134.92	1083.51
	Cosine Similarity (Window)	3411.47	1934.16	1740.91
	Weight Subcloning	2422.72	2134.92	1083.51
	Shortened Llama	5241.76	3595.71	1953.14

C UNSTRUCTURED SPARSITY

In addition, we examine performance for *unstructured sparsity*, which offers more fine-grained compression. The standard approach is to allocate sparsity *uniformly across layers*. However, some layers may be more sensitive to sparsity, which can significantly impact the model's output. To address this, OWL (Yin et al., 2024) introduces the Layer Outlier Distribution (LOD) metric as a measure of layer saliency, and computes a sparsity profile that is weighted by LOD. We compare EvoPress with both uniform sparsity and OWL. For OWL we used the same hyperparameter grid as the original work and took the configuration yielding the best perplexity for each model.

Search Space. Sparsity levels are generated as follows: For each layer, we first produce the base level corresponding to the targeted average sparsity. Then, we generate both higher and lower compression levels, where the difference between two levels corresponds to a fixed number of weights. In our experiments, we used a "step size" of 1M weights uniformly. This approach enables the mutation of compression levels across all layers, independently of their size. We adopt SparseGPT (Frantar & Alistarh, 2023) for layer pruning.

Experimental Results. We compare different methods for pruning to 50%, 60% and 70% unstructured sparsity. We report the results for 70% , 60% and 50% sparsity in Tables 8, 7 and 6, respectively. As illustrated in Table 6, EvoPress successfully finds better profiles than uniform sparsity and noticeably outperforms competitive methods on PPL and zero-shot average accuracy by large margins on all models.

Table 6: Performance of various methods at 70% average sparsity.

Model	Method	Wiki2↓	C4↓	ArcC↑	ArcE↑	HS↑	PiQA↑	WG↑	Avg↑
Mistral-7B-v0.3	Dense	4.82	7.72	48.9	79.6	60.9	80.3	73.9	68.7
	Uniform	5.68	8.93	43.7	76.7	55.7	78.4	71.0	65.1
	OWL	5.69	8.94	43.9	76.9	55.4	78.5	70.3	65.0
	EvoPress	**5.49**	**8.70**	45.7	77.3	56.5	78.9	71.2	**65.9**
Llama-2-7B	Dense	5.12	6.93	43.4	76.3	57.1	78.1	69.0	64.8
	Uniform	6.40	8.87	41.3	73.4	52.8	75.7	68.8	62.4
	OWL	6.38	8.77	41.1	73.2	53.2	76.5	70.2	62.9
	EvoPress	**6.22**	**8.52**	41.5	74.2	54.0	76.7	69.6	**63.2**
Llama-3-8B	Dense	5.54	7.10	50.4	80.1	60.2	79.7	72.6	68.6
	Uniform	8.05	13.07	43.6	75.7	54.2	76.1	71.7	64.3
	OWL	8.13	13.12	43.8	75.8	54.0	75.7	72.2	64.3
	EvoPress	**7.63**	**12.53**	43.9	77.5	54.5	76.8	72.2	**65.0**
Llama-3.1-8B	Dense	5.61	8.90	51.2	81.4	60.0	80.1	73.9	69.3
	Uniform	8.06	13.03	44.5	76.7	54.0	76.7	71.5	64.7
	OWL	8.02	12.99	44.2	76.5	53.8	76.8	72.5	64.8
	EvoPress	**7.51**	**12.31**	46.6	77.7	54.9	77.6	71.7	**65.7**

Table 7: Performance of various sparsity profiles at 60% sparsity.

Model	Method	Wiki2↓	C4↓	ArcC↑	ArcE↑	HS↑	PiQA↑	WG↑	Avg↑
Mistral-7B-v0.3	Dense	4.82	7.72	48.9	79.6	60.9	80.3	73.9	68.7
	Uniform	7.78	11.86	38.0	72.3	49.4	75.0	69.3	60.9
	OWL	7.50	11.34	38.5	71.9	49.6	75.1	70.2	61.1
	EvoPress	**7.08**	**10.27**	40.5	72.8	51.9	76.9	68.8	**62.2**
Llama-2-7B	Dense	5.12	6.93	43.4	76.3	57.1	78.1	69.0	64.8
	Uniform	9.3	12.37	35.8	69.5	45.9	72.4	65.9	57.9
	OWL	8.35	11.00	36.0	69.1	47.5	73.2	66.2	58.4
	EvoPress	**8.21**	**10.34**	37.1	70.6	49.3	74.4	67.6	**59.8**
Llama-3-8B	Dense	5.54	7.10	50.4	80.1	60.2	79.7	72.6	68.6
	Uniform	13.86	21.43	35.2	69.7	45.6	72.2	68.0	58.2
	OWL	12.37	18.53	38.0	70.3	47.7	72.1	68.5	59.3
	EvoPress	**11.02**	**16.37**	39.0	71.9	48.6	74.0	69.1	**60.5**
Llama-3.1-8B	Dense	5.61	8.90	51.2	81.4	60.0	80.1	73.9	69.3
	Uniform	13.43	21.46	36.4	69.7	46.2	72.3	67.7	58.5
	OWL	12.08	18.25	38.9	71.1	47.7	73.1	68.8	59.9
	EvoPress	**10.58**	**15.96**	40.0	72.5	49.0	74.6	69.5	**61.1**

Table 8: Performance of various sparsity profiles at 50% sparsity.

Model	Method	Wiki2↓	C4↓	ArcC↑	ArcE↑	HS↑	PiQA↑	WG↑	Avg↑
Mistral-7B-v0.3	Dense	4.82	7.72	48.9	79.6	60.9	80.3	73.9	68.7
	Uniform	5.68	8.93	43.7	76.7	55.7	78.4	71.0	65.1
	OWL	5.69	8.94	43.9	76.9	55.4	78.5	70.3	65.0
	EvoPress	**5.49**	**8.70**	45.7	77.3	56.5	78.9	71.2	**65.9**
Llama-2-7B	Dense	5.12	6.93	43.4	76.3	57.1	78.1	69.0	64.8
	Uniform	6.40	8.87	41.3	73.4	52.8	75.7	68.8	62.4
	OWL	6.38	8.77	41.1	73.2	53.2	76.5	70.2	62.9
	EvoPress	**6.22**	**8.52**	41.5	74.2	54.0	76.7	69.6	**63.2**
Llama-3-8B	Dense	5.54	7.10	50.4	80.1	60.2	79.7	72.6	68.6
	Uniform	8.05	13.07	43.6	75.7	54.2	76.1	71.7	64.3
	OWL	8.13	13.12	43.8	75.8	54.0	75.7	72.2	64.3
	EvoPress	**7.63**	**12.53**	43.9	77.5	54.5	76.8	72.2	**65.0**
Llama-3.1-8B	Dense	5.61	8.90	51.2	81.4	60.0	80.1	73.9	69.3
	Uniform	8.06	13.03	44.5	76.7	54.0	76.7	71.5	64.7
	OWL	8.02	12.99	44.2	76.5	53.8	76.8	72.5	64.8
	EvoPress	**7.51**	**12.31**	46.6	77.7	54.9	77.6	71.7	**65.7**

D REMOVAL ORDER FOR DIFFERENT METHODS

Prior research indicates that deeper layers, aside from the final ones, are generally less effective (Gromov et al., 2024; Men et al., 2024). Figure 3 illustrates the optimal removal configurations identified by EvoPress. For comparison, Table 9 displays the removal order of prior scoring methods. While EvoPress indeed removes some deeper layers across all sparsities, we also observe that certain shallow layers appear to be less important. Notably, a "two hills" pattern emerges in many cases, where blocks before and after the midpoint are pruned, yet the central blocks remain intact. Meanwhile, the first two blocks are never pruned. However, in contrast to a heuristic proposed by Ma et al. (2023), we find that, in some instances, it is effective to prune the final block as well.

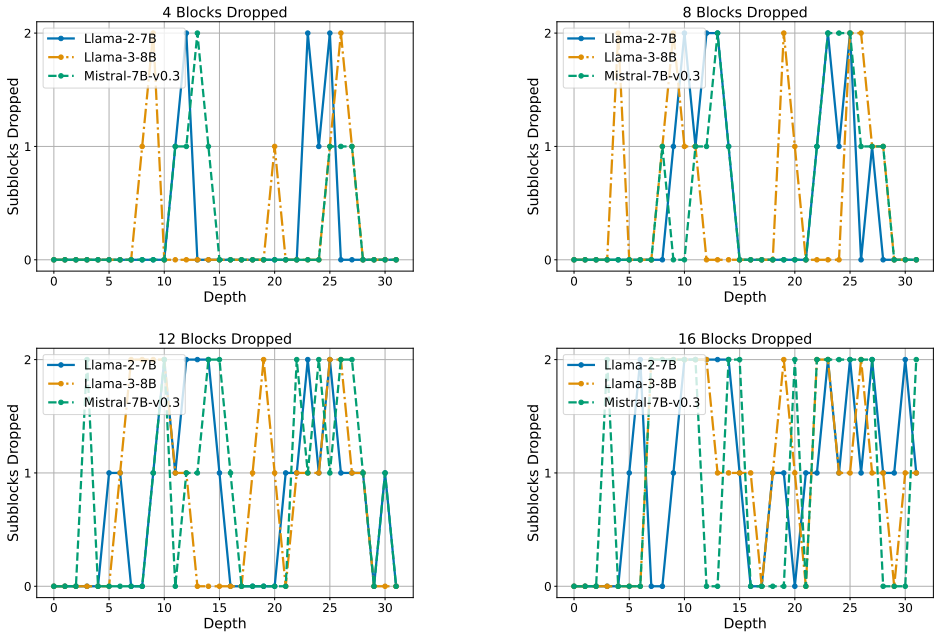

Figure 3: Optimal removal configurations identified by EvoPress for different models.

Table 9: First 16 blocks in removal order of ShortGPT, Weight Subcloning and Shortened Llama on different models.

Model	Method	Removal Order (Left to Right)
Mistral-7B-v0.3	ShortGPT	26, 25, 24, 27, 23, 22, 28, 30, 21, 29, 20, 19, 13, 17, 18, 12
	Weight Subcloning	26, 25, 24, 27, 23, 28, 22, 30, 21, 29, 20, 19, 13, 17, 12, 18
	Shortened Llama	10, 12, 13, 11, 08, 09, 14, 15, 07, 06, 04, 27, 24, 16, 25, 05
Llama-2-7B	ShortGPT	27, 25, 26, 28, 29, 24, 23, 22, 21, 30, 20, 19, 18, 17, 15, 14
	Weight Subcloning	27, 25, 28, 29, 26, 24, 23, 22, 21, 19, 30, 20, 18, 17, 14, 15
	Shortened Llama	11, 12, 08, 09, 10, 06, 24, 25, 07, 14, 23, 13, 22, 21, 15, 27
Llama-3-8B	ShortGPT	25, 26, 27, 24, 28, 23, 22, 29, 20, 21, 19, 18, 30, 17, 16, 11
	Weight Subcloning	25, 27, 26, 24, 28, 23, 22, 29, 20, 21, 19, 18, 30, 17, 16, 11
	Shortened Llama	10, 08, 09, 11, 26, 25, 12, 22, 24, 23, 14, 13, 28, 06, 19, 21
Llama-3.1-8B	ShortGPT	25, 26, 24, 27, 23, 28, 22, 29, 20, 21, 19, 18, 17, 30, 16, 10
	Weight Subcloning	25, 27, 26, 24, 28, 23, 22, 29, 20, 21, 19, 18, 30, 17, 16, 10
	Shortened Llama	10, 09, 11, 08, 26, 25, 12, 24, 22, 23, 14, 28, 06, 13, 19, 21