2SSP: A TWO-STAGE FRAMEWORK FOR STRUCTURED PRUNING OF LLMS

Fabrizio Sandri*, **Elia Cunegatti*** **& Giovanni Iacca**
Department of Information Engineering and Computer Science
University of Trento, Italy
sandri.fabr@gmail.com, {elia.cunegatti, giovanni.iacca}@unitn.it
*Equal contribution

ABSTRACT

We propose a novel Two-Stage framework for Structured Pruning (2SSP) for pruning Large Language Models (LLMs), which combines two different strategies of pruning, namely Width and Depth Pruning. The first stage (Width Pruning) removes entire neurons, hence their corresponding rows and columns, aiming to preserve the connectivity among the pruned structures in the intermediate state of the Feed-Forward Networks in each Transformer block. The second stage (Depth Pruning), instead, removes entire Attention submodules. We also propose a novel mechanism to balance the sparsity rate of the two stages w.r.t. to the desired global sparsity. We test 2SSP on four LLM families and three sparsity rates (25%, 37.5%, and 50%), measuring the resulting perplexity over three language modeling datasets as well as the performance over six downstream tasks. Our method consistently outperforms five state-of-the-art competitors over three language modeling and six downstream tasks, with an up to two-order-of-magnitude gain in terms of pruning time. The code is available at available at https://github.com/FabrizioSandri/2SSP.

1 INTRODUCTION

The sheer size of the recent, billion-scale Large Language Models (LLMs) is one of the main reasons for their successful performance. However, it comes at the cost of computational budget in terms of required GPUs as well as time for pre-training and inference, which in turn has serious economic and environmental impacts. Therefore, studying approaches to reduce the computational burden of such models while minimizing their performance degradation has become a pressing matter.

One of the main approaches to address this issue is through Network Pruning Frantar & Alistarh (2023); Ma et al. (2023), which mainly focuses on reducing the size of pre-trained LLMs as well as their inference time. Among the several pruning methods available in the literature, reliable inference speed-ups Kurtic et al. (2023); Ashkboos et al. (2024) have been achieved mainly through *structured* pruning, i.e., approaches that remove entire portions of the model. Different strategies to select which portions of the network to remove have been proposed, see 1, identifying *Width pruning*, which removes rows/columns and/or single layers, *Depth Pruning (Blocks)*, which removes entire Transformer Blocks, and *Depth Pruning (Submodules)*, which removes entire submodules (i.e., the Attention submodule–in the following, referred to as "Attention", for brevity— and/or Feed-Forward Network (FFN)) from the Transformer Blocks. *width* pruning has the main advantage of having a lower granularity level in the removal search space, which leads to a more refined identification of unimportant components of the model. On the other hand, the advantage of *depth* pruning lies in the lower computation time required to obtain the sparse structure as well as the larger inference speed-up that comes from the removal of entire blocks/submodules.

Contributions So far, the aforementioned categories of structured pruning have been investigated independently of one another and their combination is currently a relatively unexplored research direction. In this paper, we propose a Two-Stage Framework for Structured Pruning (2SSP), a new structured pruning approach that, to our knowledge, is the first to combine *width* and *depth* pruning, hence exploiting the advantages of both approaches. The proposed 2SSP works in two stages. The

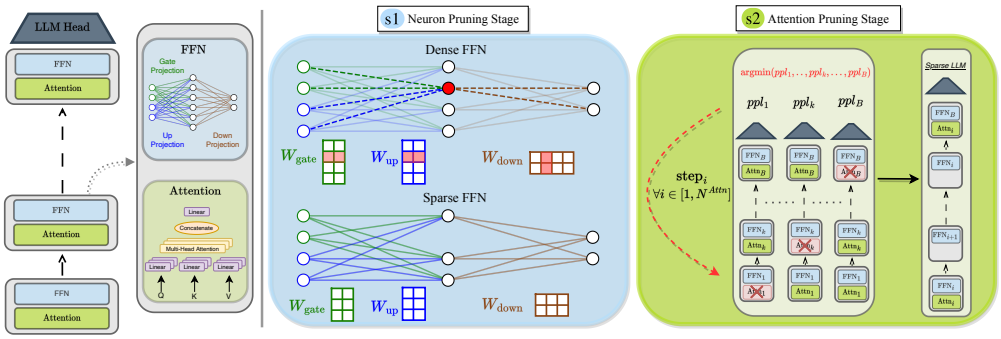

Figure 1: Conceptual scheme of 2SSP. The left part shows the original LLM; the central part indicates stage s1, which focuses on FFNs; the right part indicates stage s2, which focuses on Attention.

first stage works at a lower granularity level, i.e. it uses *width* pruning to remove neurons from the intermediate state of FFNs based on the output's magnitude. This is done while preserving the network connectivity, which is a critical measure for reducing performance degradation in sparse structures Vysogorets & Kempe (2023); Cunegatti et al. (2024); Iurada et al. (2024). Moreover, the first stage is applied only to the FFN parts of the model, for which pruning is known to be more difficult than Attentions Siddiqui et al. (2024). The second stage complements the effect of the first one by iteratively applying *depth pruning* on Attentions, based on the minimization of a given performance metric (in our case, perplexity).

2 METHODS

We now introduce 2SSP, a novel pruning framework that combines two stages of structured pruning: a first stage (s1) which prunes at the level of entire neurons by removing rows and columns from the FFN within the Transformer blocks; and a second stage (s2), which performs submodule-based depth pruning by removing entire Attentions.

Neuron Pruning (s1) The method aims to prune the neurons of the intermediate representation within the Linear layers of the FFN following the Attention in each Transformer block. The rationale is that the FFN's hidden state generates an intermediate representation that can be compressed by removing entire neurons from its hidden state, obtaining a lower-dimensional representation that preserves the most important features of the input sequence. The algorithm prunes an equal number of neurons from each FFN, removing the neurons in the hidden state that have the lowest impact, measured as the magnitude of their activated output. The magnitude is calculated as the average $\mathcal{L}_2$ norm across the tokens in a set of calibration samples from a given calibration dataset $\mathcal{D}_{\text{cal}}$. The top-K neurons are then retained in the FFN of each block.

Let the intermediate representation of the FFN in block b be denoted as $\mathbf{Z}_b \in \mathbb{R}^{T \times d_{\text{int}}}$, where T is the sequence length and d_{int} is the dimension of the intermediate representation of the FFN in block b. For each neuron j, we compute an importance score s_j across the calibration dataset $\mathcal{D}_{\text{cal}}$:

$$s_j = \frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{c=1}^{|\mathcal{D}_{\text{cal}}|} \|z_c^{(j)}\|_2 \tag{1}$$

where $z_c^{(j)}$ is the activation of the j-th neuron for the c-th sequence in the calibration dataset, and $\|\cdot\|_2$ denotes the $\mathcal{L}_2$ norm over the tokens in the calibration sequence.

Formally, let $\mathbf{W}_{\text{in}} \in \mathbb{R}^{d_{\text{int}} \times d_{\text{model}}}$ and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{int}}}$ be the input and output projection matrices of the FFN, respectively, where d_{model} is the model's hidden dimension. We define a binary mask $\mathbf{m} \in \{0,1\}^{d_{\text{int}}}$ that selects the top-K neurons to preserve based on their importance scores s_j. The pruned weight matrices are then computed as: $\hat{\mathbf{W}}_{\text{in}} = \mathbf{W}_{\text{in}}[\mathbf{m} = 1, :]$, and $\hat{\mathbf{W}}_{\text{out}} = \mathbf{W}_{\text{out}}[:, \mathbf{m} = 1]$.

To illustrate the mechanism, let us consider a simple FFN with two Linear layers, denoted as l_{in} and l_{out}. Removing a neuron from the hidden state of the FFN necessitates the removal of all the associated weights in both l_{in} and l_{out} connected to that particular neuron. More precisely, given the weight matrices $\mathbf{W}_{\text{in}} \in \mathbb{R}^{d_{\text{int}} \times d_{\text{model}}}$ and $\mathbf{W}_{\text{out}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{int}}}$, respectively of l_{in} and l_{out}, the pruning

of a hidden state neuron involves eliminating (in this order, to preserve network connectivity) the corresponding row in $\mathbf{W}_{\text{in}}$ and the associated column in $\mathbf{W}_{\text{out}}$.

Attention Pruning (s2) Pruning only the FFN submodules limits the effectiveness of the algorithm, as only a restricted fraction of the total number of parameters can be pruned leaving all the Attention parameters intact. To address this limitation, inspired from the observation derived in Siddiqui et al. (2024) where it has been shown how the Attentions can be removed to a certain degree ($\sim 33\%$) with almost no performance degradation, we propose a second pruning stage that, after removing a certain fraction of FFN neurons, also prunes the remaining parameters from the Attentions. Unlike FFNs, Attentions do not create a single hidden state due to the sequential application of multiple mathematical operators, such as scaling, softmax, and matrix multiplication. This makes pruning entire neurons impractical. We address this challenge by adopting the submodules pruning mechanism proposed in Zhong et al. (2024), by removing only Attentions (and not also FFNs as in the original mechanism) from the sparse network $\mathcal{M}_1$ obtained after the first stage. Specifically, we iteratively remove the Attentions leading to the lowest perplexity on the calibration dataset until the target sparsity is reached.

Formally, let $\mathcal{A} = \{a_1, \ldots, a_B\}$ be the set of Attentions across all the B blocks of the Transformer. At each step t, we select the Attention module a^* whose removal minimizes the perplexity on the calibration dataset: $a^* = \underset{a \in \mathcal{A}_t}{\text{argmin}}\, \text{PPL}(\mathcal{M}_t \setminus a, \mathcal{D}_{\text{cal}})$, where $\mathcal{A}_t$ is the set of the remaining Attention modules at step t, $\mathcal{M}_t \setminus a$ denotes the model at step t with the Attention module a removed, and PPL represents the perplexity metric, calculated as the exponential of the negated average log-likelihood over the calibration samples of $\mathcal{D}_{\text{cal}}$.

Balancing the Sparsity Rate As explained above, the proposed 2SSP algorithm works in two stages. Hence, given a target sparsity rate s, selecting the pruning rate allocated for each of the two stages is also required. Through empirical analysis, we found out that the following equation provides a reliable metric for determining the optimal number of Attention modules to prune at any given sparsity rate: $N^{\text{Attn}} = \text{round}\left(B \cdot s^{\frac{|W_{\text{FFN}}|}{\alpha |W_{\text{Attn}}|}}\right)$, where $\alpha = 1.5$, $|W_{\text{FFN}}|$ represents the number of FFN parameters per block, and $|W_{\text{Attn}}|$ represents the number of Attention parameters per block. This equation captures two critical aspects of this pruning process. First, it ensures that the number of pruned Attention parameters scales with increasing sparsity rates. Second, it adjusts the Attention pruning rate based on the relative sizes of the FFN and Attention modules. Specifically, when $\frac{|W_{\text{FFN}}|}{|W_{\text{Attn}}|}$ is large, indicating that FFN parameters dominate the block structure, the equation reduces the proportion of Attention parameters to be pruned. This adaptive behavior helps maintain the balance between the number of Attentions and the number of FFN parameters pruned.

3 EXPERIMENTS

In this section, we evaluate 2SSP in terms of performance over both language modeling and downstream tasks. w.r.t. the selected baselines: namely ShortGPT Ma et al. (2023), and Sliding Window Gromov et al. (2024) for the *Depth Block* category, BlockPruner Zhong et al. (2024), and EvoPress Sieberling et al. (2024) *Depth Submodule* category, and SliceGPT Ashkboos et al. (2024) for the *Depth Width*. 2 reports all the numerical results in terms of perplexity across the tested dataset and sparsity rates. It is clear how our proposed approach outperforms the baselines in all test cases. Even more interesting is the robustness of our approach w.r.t. both models and sparsity rates.

We also tested if our approach could still outperform all the baselines at sparsity rates different from 25%, 37.5%, and 50%. For doing so, we

Table 1: Zero-shot performance for 2SSP vs. the compared pruning models at 37.5% sparsity.

Model	Algorithm	MMLU	WQ	PIQA	HS	ARC-e	ARC-c	Average
Mistral-v0.3 7B	Dense	59.08	73.72	80.30	60.91	79.67	48.81	67.08
	ShortGPT	22.67	58.56	56.96	27.73	33.59	**29.27**	38.13
	Sliding Window	**25.54**	57.22	59.19	29.41	34.51	26.88	38.79
	BlockPruner	23.59	54.62	66.16	37.92	46.25	24.32	42.14
	EvoPress	25.00	57.14	68.82	39.79	**50.21**	25.94	44.48
	SliceGPT	23.15	61.88	65.34	36.95	42.68	21.42	41.90
	2SSP	24.49	**63.14**	**70.29**	**41.99**	49.96	24.49	**45.73**
Llama-2 7B	Dense	40.67	68.90	78.07	57.09	76.22	43.34	60.72
	ShortGPT	32.25	60.54	59.63	33.54	41.33	**28.50**	42.63
	Sliding Window	**33.38**	58.64	60.07	33.47	36.15	28.41	41.69
	BlockPruner	23.59	55.09	66.87	36.92	50.80	24.49	42.96
	EvoPress	25.66	52.01	68.61	37.15	53.20	25.94	43.76
	SliceGPT	23.07	**63.85**	67.90	40.40	47.56	26.19	44.83
	2SSP	27.91	61.33	**70.29**	**42.78**	**55.93**	27.39	**47.61**
Qwen-2.5 7B	Dense	71.88	73.24	78.56	59.97	80.35	48.29	68.72
	ShortGPT	23.02	51.46	63.98	33.90	50.55	24.74	41.27
	Sliding Window	23.76	52.49	65.02	34.36	54.46	23.12	42.20
	BlockPruner	**25.15**	53.35	67.41	36.97	58.59	**27.22**	44.78
	EvoPress	24.21	55.17	67.30	36.95	**59.76**	27.13	45.09
	SliceGPT	22.91	57.70	65.94	34.32	48.02	20.48	41.56
	2SSP	23.34	**61.40**	**70.29**	**43.75**	52.65	26.79	**46.37**
Phi-3 14B	Dense	67.61	75.77	81.01	64.04	84.05	60.67	72.19
	ShortGPT	27.03	52.09	56.04	27.15	34.34	28.50	37.53
	Sliding Window	25.17	50.04	52.77	25.65	26.22	23.04	33.82
	BlockPruner	27.90	61.40	68.12	42.00	62.75	37.37	49.93
	EvoPress	34.63	60.14	67.85	41.46	61.74	35.58	50.23
	SliceGPT	27.21	66.61	71.16	45.45	54.34	29.78	49.09
	2SSP	**51.85**	**68.82**	**74.97**	**51.60**	**67.26**	**38.99**	**58.92**

Table 2: Perplexity for 2SSP vs. the compared pruning algorithms over three different sparsity rates across four LLMs. The boldface and underline indicate, respectively, the best and second-best value per dataset (excluding the dense baseline).

Sparsity	Method	Mistral-v0.3 7B			LLama-2 7B			Qwen-2.5 7B			Phi-3 14B		
		WikiText2	C4	Fineweb-Edu	WikiText2	C4	Fineweb-Edu	WikiText2	C4	Fineweb-Edu	WikiText2	C4	Fineweb-Edu
0%	Dense	5.36	8.13	6.49	5.47	7.13	6.44	6.85	11.68	7.7 2	4.31	8.54	6.41
25%	ShortGPT	44.65	38.62	32.81	25.43	31.04	22.8	13.09	19.38	14.24	129.79	124.02	139.66
	Sliding Window	37.75	52.26	42.49	18.25	21.71	17.95	11.37	17.54	12.75	31.13	30.28	24.39
	BlockPruner	10.33	13.43	10.95	12.09	12.98	11.04	11.57	17.43	12.33	9.76	13.16	9.83
	EvoPress	9.35	12.86	10.39	10.37	11.92	10.23	11.74	17.29	12.41	8.71	12.34	9.53
	SliceGPT	11.84	13.09	11.36	14.82	12.57	11.19	12.72	17.40	13.88	10.01	11.86	9.82
	2SSP (Ours)	**9.24**	**12.19**	**10.27**	**9.25**	**10.52**	**9.21**	**10.61**	**15.67**	**11.92**	**7.06**	**10.43**	**8.42**
37.5%	ShortGPT	$1.83e^3$	$1.49e^3$	$1.31e^3$	79.49	66.69	54.07	52.99	48.07	36.98	$1.34e^5$	$1.33e^5$	$1.48e^5$
	Sliding Window	984.31	$1.40e^3$	$1.36e^3$	207.04	225.83	172.21	21.73	30.88	23.07	$1.20e^6$	$5.71e^5$	$5.03e^5$
	BlockPruner	23.31	25.66	23.23	23.62	21.47	18.13	22.17	29.49	21.76	19.31	21.81	17.15
	EvoPress	27.00	24.10	19.93	19.03	20.22	17.04	22.03	28.98	21.79	15.62	19.89	15.18
	SliceGPT	23.24	21.41	19.98	30.28	19.58	18.71	25.08	28.07	25.31	17.06	15.81	14.52
	2SSP (Ours)	**14.92**	**17.15**	**15.03**	**14.64**	**14.93**	**13.36**	**15.26**	**20.95**	**16.89**	**9.79**	**12.88**	**10.91**
50%	ShortGPT	$1.01e^3$	963.5	889.31	233.18	187.46	160.98	$9.30e^4$	$1.27e^8$	$3.63e^8$	$4.40e^5$	$2.79e^5$	$3.67e^5$
	Sliding Window	$3.31e^3$	$1.56e^3$	$1.82e^3$	$3.34e^3$	$2.48e^3$	$2.42e^3$	213.96	177.08	142.71	$1.01e^6$	$1.08e^6$	$9.97e^5$
	BlockPruner	81.90	64.85	54.64	71.36	55.46	48.10	54.97	63.00	47.94	56.6	57.36	46.40
	EvoPress	91.83	73.71	60.55	70.97	44.39	38.58	49.91	57.95	43.45	54.46	50.22	39.95
	SliceGPT	41.24	35.29	33.96	57.66	36.06	35.55	38.47	41.98	37.56	34.40	26.42	27.21
	2SSP (Ours)	**23.77**	**25.95**	**23.30**	**31.40**	**27.16**	**25.40**	**21.66**	**28.00**	**23.72**	**16.93**	**18.82**	**17.07**

computed the perplexity over WikiText2 for different sparsity rates ranging from zero to 70%. The results for Mistral-v0.3 7B and LLama-2 7B are shown in 2, where the performance trend of 2SSP w.r.t. the baselines indicated in 2 is confirmed even across such a broader set of sparsity rates. In order to assess the performance of our approach not only in terms of perplexity, we considered six different downstream tasks. In this case, we tested 2SSP, as well as the baselines, at 37.5% sparsity (the intermediate value among those considered) considering the average zero-shot accuracy as the main metric. Table 1 shows the results of this experimental setting, showing once again how 2SSP outperforms the tested baselines in most of the cases (and always on average across all tasks).

Pruning Runtime For any pruning algorithm to be considered effective, along with the performance obtained by the generated sparse model, also the time required to obtain such sparse models is a critical metric. For this reason, we compare the performance (in terms of perplexity over WikiText2) vs. the pruning runtime of 2SSP and the baselines. 3 shows the trade-off between these two metrics for the three sparsity rates tested in 2, for the case LLama-2 7B. The results clearly show how 2SSP is the best algorithm in terms of performance vs. pruning runtime trade-off. As expected, the fastest algorithms are the ones belonging to the *Depth Pruning (Blocks)* category, since they apply pruning in one-shot, w.r.t. the similarity among blocks. On the other hand, the algorithms from the *Depth Pruning (Submodules)* category are the slowest ones, since they evaluate each possible submodule removal combination. To conclude, our approach, which is a combination of one-shot removal (s1) and submodule evaluation (s2) can achieve the best performance in terms of perplexity while requiring limited pruning runtime.

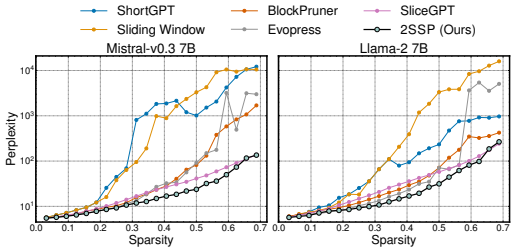

Figure 2: Perplexity for 2SSP vs. the compared pruning algorithms for $s \in [0, 0.7]$.

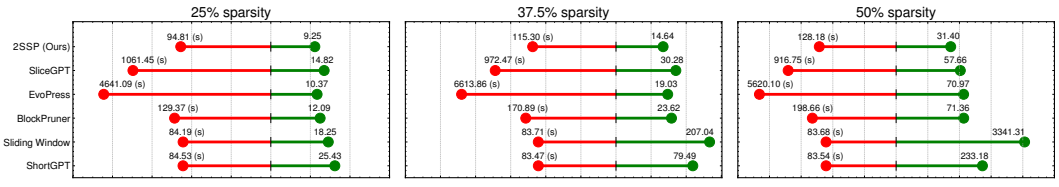

Figure 3: Pruning runtime (red, left side of the x-axis, log scale) vs. perplexity (green, right side of the x-axis, log scale) for 2SSP vs. the compared pruning algorithms over LLama-2 7B.

4 Conclusions

In this paper, we introduced 2SSP, a new structured pruning algorithm that aims to combine *Width Pruning* for FFN submodules with *Depth Pruning* for Attentions. Our approach works in two stages

by firstly pruning neurons in the intermediate state of FFN submodules, and then iteratively removing Attentions based on the model performance computed as perplexity. The results demonstrate how our proposed algorithm consistently outperforms the state-of-the-art baselines on both language modeling and downstream tasks. 2SSP achieves these results while requiring limited pruning runtime, which positions our method as state-of-the-art over the performance vs. pruning runtime trade-off.

REFERENCES

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress Large Language Models by Deleting Rows and Columns. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=vXxardq6db.

Elia Cunegatti, Matteo Farina, Doina Bucur, and Giovanni Iacca. Understanding Sparse Neural Networks from their Topology via Multipartite Graph Representations. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=Egb0tUZnOY.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023. URL https://dl.acm.org/doi/10.5555/3618408.3618822.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

Leonardo Iurada, Marco Ciccone, and Tatiana Tommasi. Finding Lottery Tickets in Vision Models via Data-driven Spectral Foresight Pruning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16142–16151, 2024.

Eldar Kurtic, Elias Frantar, and Dan Alistarh. ZipLM: Inference-Aware Structured Pruning of Language Models. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=d8j3lsBWpV.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 21702–21720, 2023. URL https://openreview.net/forum?id=J8Ajf9WfXP.

Shoaib Ahmed Siddiqui, Xin Dong, Greg Heinrich, Thomas Breuel, Jan Kautz, David Krueger, and Pavlo Molchanov. A deeper look at depth pruning of LLMs. In *ICML 2024 Workshop on Theoretical Foundations of Foundation Models*, 2024. URL https://openreview.net/forum?id=9B7ayWclwN.

Oliver Sieberling, Denis Kuznedelev, Eldar Kurtic, and Dan Alistarh. Evopress: Towards optimal dynamic model compression via evolutionary search. *arXiv preprint arXiv:2410.14649*, 2024.

Artem Vysogorets and Julia Kempe. Connectivity matters: Neural network pruning through the lens of effective sparsity. *Journal of Machine Learning Research*, 24(99):1–23, 2023.

Longguang Zhong, Fanqi Wan, Ruijun Chen, Xiaojun Quan, and Liangzhi Li. Blockpruner: Fine-grained pruning for large language models. *arXiv preprint arXiv:2406.10594*, 2024.