OPT2025: 17th Annual Workshop on Optimization for Machine Learning

Simultaneous Fine-Tuning and Pruning of LLMs

Finn Reinecke REINECFI@GMAILCOM
Jörg Franke FRANKEJ@CS.UNI-FREIBURG.DE
Frank Hutter FH@CS.UNI-FREIBURG.DE
Michael Hefenbrock MICHAEL.HEFENBROCK@PERSPIX.AI

Abstract

Fine-tuning large language models (LLMs) for specific downstream tasks enables exceptional performance; unfortunately, the vast model sizes hinder deployment in hardware-constrained environments. Hence, small, domain-specific models are created for such scenarios. This is usually done in a two-stage process by first pruning a LLM and fine-tuning (FT) afterwards. However, performing these two steps jointly may yield better results, as both FT and pruning can then adapt to each other. Motivated by this potential, we propose a method based on constrained optimization that uses augmented Lagrangian methods to simultaneously fine-tune and prune (SFP) LLMs to a target sparsity. Our approach is directly compatible with parameter-efficient fine-tuning (PEFT) techniques and can be applied to structures of different granularities. We evaluate the effectiveness of the method against state-of-the-art pruning techniques and show similar or better performance. Specifically, SFP can prune a 7 billion parameter model to 50% sparsity and achieve a 1.88 times faster inference speed with negligible performance degradation.

1. Introduction

LLMs achieved great capabilities over a variety of tasks by scaling up auto-regressive language models [15]. In the current paradigm, models are pre-trained on a large and general corpus of data and then adapted to downstream tasks with PEFT techniques on a smaller corpus of domain-specific data [16].

Although PEFT enables a reduction in resource requirements during training, there is no reduction in inference costs. This can hinder practical usage do to the huge model sizes. To reduce hardware demand, model compression techniques like pruning have been explored, see e.g. [31]. We identify three pathways to create a small and task-specific model from a large pretrained one: 1) pruning first and fine-tuning afterwards, 2) vice versa, or 3) simultaneous fine-tuning and pruning. Those pathways are depicted in Figure 1. Separating FT and pruning into two distinct stages adds complexity and disregards the possibility of guiding the FT process by pruning decisions. Consequently, simultaneously finetuning and pruning should be the most promising approach and thus motivates this work.

Our core contributions is the theoretical derivation and evaluation of a new structured pruning method that allows for simultaneous fine-tuning and pruning to a desired sparsity. The proposed method employs constrained optimization techniques to enforce the desired sparsity level on the

© F. Reinecke, J. Franke, F. Hutter & M. Hefenbrock.

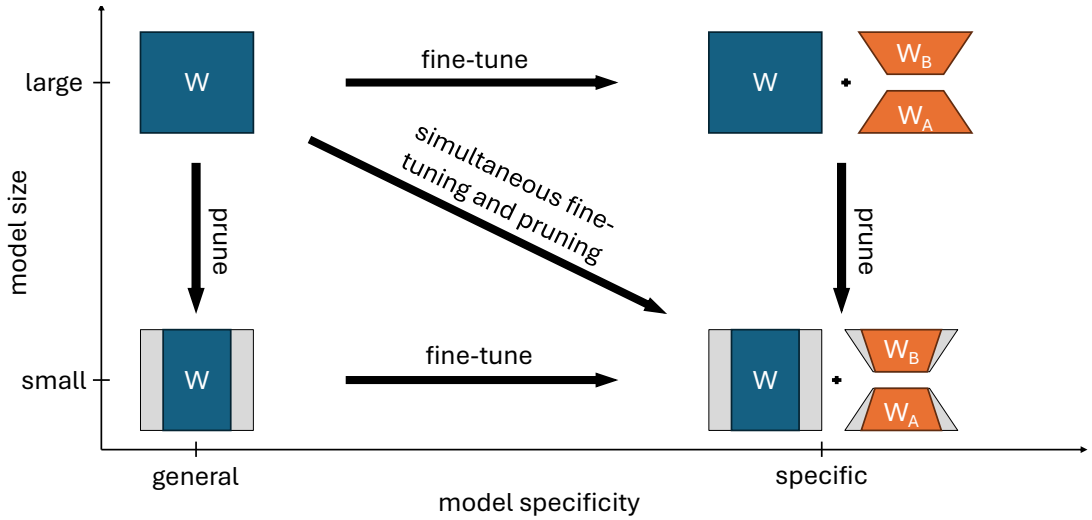

Figure 1: In today's paradigm there are large but very capable general pre-trained models. The pathway to a small model for a specific task is typically a two stage process of pruning and fine-tuning (pathway 1) or vice versa (pathway 2). Combining those two stages into one gives pathway 3.

model during the fine-tuning process. It can be seamlessly integrated with existing optimizers and PEFT techniques.

In our experiments, we compare our proposed method to SoTA baseline methods on different sparsity levels and model sizes. The baselines consist of random pruning, a one-stage pruning and fine-tuning (pathway 3) approach, and a traditional two-stage method (pathway 1). All evaluations are done for the task of creating and compressing a guard for human-LLM interaction. Since such guards are available at different sizes, efficient sparsification of these guards is an important consideration. In particular, to ensure reliable safety for AI even under resource constraints.

2. Related Work

Parameter Efficient Fine-Tuning PEFT techniques [16, 18, 20] have become a popular way to FT pre-trained models with less hardware requirements than full-FT. Low-Rank Adaptation, for example (LoRA), introduces trainable adapter matrices of low rank which are used to update layers [16]. LoRA and LoRA-based methods like QLoRA or LongLoRA [5, 6] reduce training requirements but do not reduce the number of active model parameters. This leaves potential for inference speedups.

Pruning In the literature, multiple ways for neural network (NN) compression have been explored. Those can be divided into quantization, knowledge distillation (KD), and pruning [31]. This work focuses on pruning, as KD needs a teacher model that, in general, does not exist before FT, and quantization is a post-training approach.

Within the pruning field, unstructured [7, 12, 13, 30], semi-structured [8, 23, 25] and structured [21, 22, 26] pruning methods have been developed. Unstructured pruning methods remove

individual weights without regular patterns. Notable representatives of this subfield include magnitude pruning (MP) [13], which removes weights below a certain threshold, and the lottery ticket hypothesis (LTH) [7]. The LTH is based on MP and introduces theoretical insights into optimal sub-networks existing in a NN since initialization.

Especially important, in the context of our work, is the surrogate Lagrangian relaxation [30], which is similar to the proposed method in an unstructured case. This whole line of work needs specialized hardware or software to enable inference speedups.

Semi-structured pruning methods remove individual weights based on regular patterns. The N:M sparsity pattern removes N out of every group of M weights [23, 31]. Notable works in this context are pruning by weights and activations (Wanda) [25], SparseGPT [8], and Outlier Weighed Layerwise sparsity (OWL) [27]. Still, semi-structured pruning methods rely on hardware that supports N:M sparsity patterns.

Structured pruning methods remove entire structures of a NN like channels, attention heads (width pruning), or entire layers (depth pruning) [19, 21, 22, 24, 31]. Prominent methods like LLM-pruner [21], which use the first-order Taylor expansion as pruning criterion, rely on gradients of the pre-trained weights and thus are not directly compatible with PEFT techniques like LoRA [28].

Recent methods that tackle the problem of simultaneous FT and pruning of LLMs rely on KD to avoid end-task performance drops [29] or use different criteria to prune structures at different granularity [10]. The probably most prominent of those methods, called LoRAprune [28], only enables width pruning.

3. Method

To combine fine-tuning with pruning to a target sparsity, we formulate fine-turning as a constrained optimization problem. The objective will be the classical fine-tuning loss, while the constraint expresses the target sparsity. Specifically, the fine-tuning problem becomes

$$\min_{\theta} \ \mathcal{L}(\theta) \quad \text{s.t.} \quad c(\theta) = L_0(\theta) - \kappa \leq 0. \tag{1}$$

Here, $\mathcal{L}(\theta)$ denotes the fine-tuning loss (e.g., cross entropy) and $c(\theta)$ is the constrained expressing the target sparsity in the form of an L_0 constraint on θ (i.e., demanding κ entries in θ to be zero).

Notably, the constraint $c(\theta)$ is not differentiable. However, by substitution of θ in the constraint with auxiliary variables $u^\star$ (derivation see Appendix A) and applying an augmented Lagrangian formulation [14], one can write this as

$$\min_{\theta} \ \mathcal{L}(\theta) + R(\theta), \tag{2}$$

with

$$R(\theta) = \lambda^\top(\theta - u^\star) + \frac{\mu}{2}\|\theta - u^\star\|^2. \tag{3}$$

The Lagrange multipliers λ are updated using

$$\lambda \leftarrow \lambda + \mu(\theta - u^\star), \tag{4}$$

where μ denotes the Lagrange multiplier update rate, which we set to $\mu = 1$ in the following. The auxiliary variables $u^\star$ are given elements wise by

$$u_i^\star = \begin{cases} \theta_i + \frac{\lambda_i}{\mu} & \text{if} \quad i \in \mathcal{Q} \\ 0 & \text{else} \quad i \notin \mathcal{Q}. \end{cases} \tag{5}$$

The set $\mathcal{Q}$ contains the indexes i relating to the top-κ largest $|\theta_i + \frac{\lambda_i}{\mu}|$. For a detailed derivation, see Appendix A. For the implementation, Equations (3) and (4) can be incorporated directly into an optimizer as shown in Algorithm 1. To enable structured pruning, similar to [17], we introduce auxiliary parameters ϕ. They are initialized to 1 and get pruned instead of the original model parameters θ. In this way, we employ two different variants of structured pruning: Channel-wise simultaneous fine-tuning and pruning (CSFP) introduces one parameter for each channel in the feed-forward network (FFN) and one for each head in the attention module of a transformer block. Module-wise simultaneous fine-tuning and pruning (MSFP) introduces one parameter for each FFN layer and one for each attention module in a transformer-based LLM. Further details see Appendix B.

Algorithm 1: Training Step with SFP

given: learning rate $\eta \in \mathbb{R}^+$, pruning learning rate $\eta_{prune} \in \mathbb{R}^+$, optimizer $\mathrm{Opt}(\cdot)$, loss func. $\mathcal{L}$
input and output data X, Y
initialize: parameters θ, step $t \leftarrow 0$, Lagrange multipliers $\lambda \leftarrow 0$
while *stopping criteria are not met* **do**
  $t \leftarrow t + 1$
  $\theta \leftarrow \theta + \mathrm{Opt}\left(\mathcal{L}(\theta, X_t, Y_t), \eta\right)$
  for *each pruned parameter group* θ_j *in* θ **do**
    Calculate $u_j^\star$ according to Equation (5).
    $\lambda_j \leftarrow \lambda_j + (\theta_j - u_j^\star)$
    $\theta_j \leftarrow \theta_j - \eta_{\text{prune}} \cdot (\lambda_j + (\theta_j - u_j^\star))$
  end
end

4. Experiments

For all our experiments, we use the Llama-2-7b-hf [1] and the WildGuard dataset [3, 11] for the task of jointly creating and pruning a guard model for human-LLM interactions. All experiments with two seeds each, are performed on either a NVIDIA A40s, NVIDIA A100-SXM4-40GBs, or NVIDIA H200s.

In preliminary experiments, we tuned the hyperparameters (HP) of the proposed methods CSFP and MSFP. We describe the corresponding experiments in Appendix D and use the optimal values in the following experiments. For all considered methods, we conducted experiments for the target sparsity levels $s = \{0.125, 0.250, 0.500, 0.750, 0.875\}$, employed a learning rate $\eta = 2.5e-4$, a batch size $bs = 80$, and used LoRA with rank $r = 48$ and scaling $\alpha = 1$. The Lagrange multipliers are only updated every $n_{\text{lag}} = 200$ steps, and the pruning learning rates $\eta_{\phi,\text{prune}}$ for the auxiliary parameters ϕ are set to 0.01 and 0.05 for CSFP and MSFP, respectively. The corresponding learning rates that the (base) optimizer for training uses ϕ is set to $\eta_{\phi,\text{opt}} = 0.01$ and $\eta_{\phi,\text{opt}} = 0.02$ for CSFP and MSFP, respectively (see also Appendix B.3).

As comparison, we use: 1) default LoRA [16] fine-tuning (no pruning), 2) the ShortGPT method based on the block influence (BI) [22], which is a two stage first prune and then FT approach, 3) the LoRAprune (LRP) [28] method which is a popular one stage approach. Figure 2 and Table 3 show the $F1$-score on the test data against the inference time t_{inf} on 20 batches (measured on a NVIDIA A40). The results with the CSFP methods are consistently worse than the other methods and only

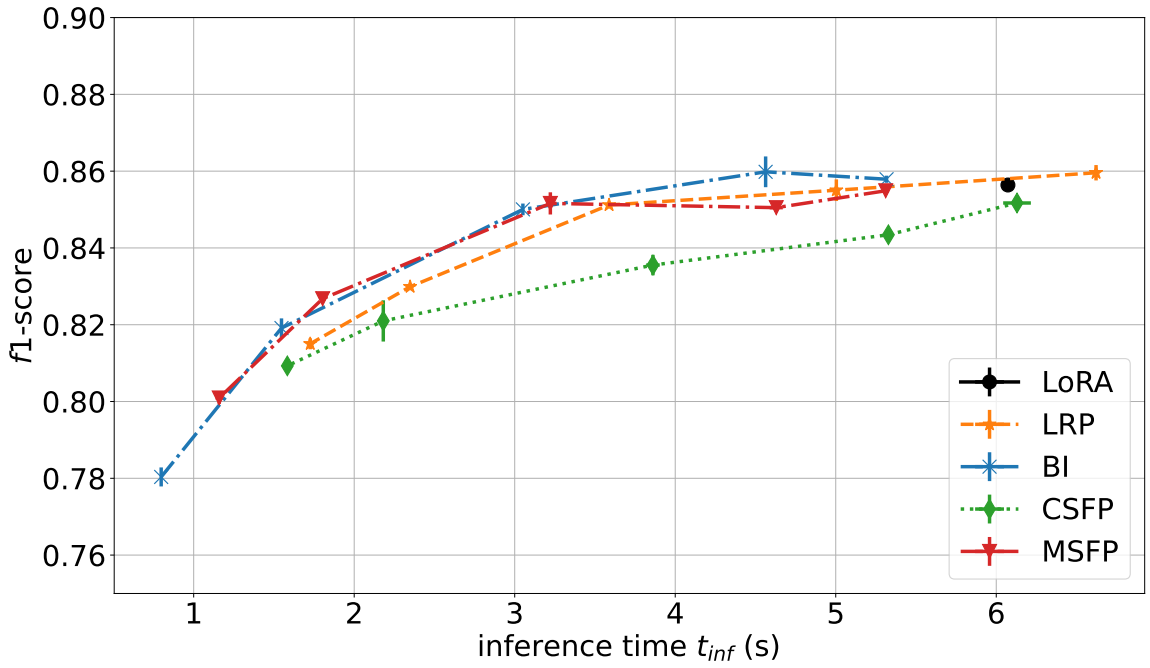

Figure 2: The $F1$-score against the inference time t_{inf} of pruned Llama-2-7b-hf models on the test set for the different baseline methods LoRA (black), LRP (orange), BI (blue) and proposed methods CSFP (green) and MSFP (red).

get close to the Pareto front at high sparsity levels. Models of different sparsities of the MSFP and the BI approach almost exclusively occupy the Pareto front in this experiment. MSFP reduces the inference time from the default LoRA $t_{\text{inf}} = 6.07$s to $t_{\text{inf}} = 3.222$, which is a 1.88 times speedup at a negligible performance drop from $F1 = 0.856$ to $F1 = 0.852$.

5. Conclusion

This work developed and characterized a new method for simultaneous fine-tuning and pruning based on the augmented Lagrangian method. It seamlessly integrates with PEFT techniques like LoRA and existing optimizers for training, making it a flexible and easy-to-use approach. Further, the method can prune structures of different granularity. The experiments show that the performance of the proposed depth-wise variant MSFP is on par or better than state-of-the-art approaches and can speed up inference time up to 1.88 times at negligible performance drop. The width pruning variant lacks performance CSFP compared to the baseline method. Overall, it is important to note that experiments were only conducted on one dataset, which limits the findings. Further experiments need to be conducted to check if the found HPs and the observed performance generalize to different tasks. Still, the results are promising, especially for the MSFP variant and the CSFP variant in the high sparsity regime. Thus, combining MSFP and CSFP for joint pruning at different structure granularities could yield further improvements.

Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG. This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. We acknowledge funding by the European Union (via ERC Consolidator Grant DeepLearning 2.0, grant no. 101045765). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] meta-llama/llama-2-7b-hf. https://huggingface.co/meta-llama/Llama-2-7b-hf, . Accessed: 13.07.2025.

[2] meta-llama/llama-3.2-1b. https://huggingface.co/meta-llama/Llama-3.2-1B, . Accessed: 13.07.2025.

[3] Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. https://huggingface.co/allenai/wildguard, . Accessed: 11.07.2025.

[4] Dimitri P Bertsekas. Approximation procedures based on the method of multipliers. *Journal of Optimization Theory and Applications*, 23(4):487–510, 1977.

[5] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *arXiv preprint arXiv:2309.12307*, 2023.

[6] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

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

[9] Jose Gallego-Posada, Juan Ramirez, Akram Erraqabi, Yoshua Bengio, and Simon Lacoste-Julien. Controlled sparsity via constrained optimization or: How i learned to stop tuning penalties and love constraints. *Advances in Neural Information Processing Systems*, 35:1253–1266, 2022.

[10] Naibin Gu, Peng Fu, Xiyu Liu, Bowen Shen, Zheng Lin, and Weiping Wang. Lightpeft: Lightening parameter-efficient fine-tuning via early pruning. *arXiv preprint arXiv:2406.03792*, 2024.

[11] Seungju Han, Kavel Rao, Allyson Ettinger, Liwei Jiang, Bill Yuchen Lin, Nathan Lambert, Yejin Choi, and Nouha Dziri. Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms. *arXiv preprint arXiv:2406.18495*, 2024.

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

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

[14] Magnus R Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

[15] 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. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[16] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1 (2):3, 2022.

[17] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. *CoRR*, abs/1707.01213, 2017. URL http://arxiv.org/abs/1707.01213.

[18] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *European conference on computer vision*, pages 709–727. Springer, 2022.

[19] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[20] Gen Luo, Minglang Huang, Yiyi Zhou, Xiaoshuai Sun, Guannan Jiang, Zhiyu Wang, and Rongrong Ji. Towards efficient visual adaption via structural re-parameterization. *arXiv preprint arXiv:2302.08106*, 2023.

[21] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

[22] 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. *arXiv preprint arXiv:2403.03853*, 2024.

[23] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[24] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. *Advances in Neural Information Processing Systems*, 37:41076–41102, 2024.

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

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

[27] Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Gen Li, Ajay Jaiswal, Mykola Pechenizkiy, Yi Liang, et al. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. *arXiv preprint arXiv:2310.05175*, 2023.

[28] Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. Loraprune: Structured pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*, 2023.

[29] Bowen Zhao, Hannaneh Hajishirzi, and Qingqing Cao. Apt: Adaptive pruning and tuning pre-trained language models for efficient training and inference. *arXiv preprint arXiv:2401.12200*, 2024.

[30] Shanglin Zhou, Mikhail A Bragin, Deniz Gurevin, Lynn Pepin, Fei Miao, and Caiwen Ding. Surrogate lagrangian relaxation: A path to retrain-free deep neural network pruning. *ACM Transactions on Design Automation of Electronic Systems*, 28(6):1–19, 2023.

[31] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12: 1556–1577, 2024.

Appendix A. Theoretical derivation

This section develops the core update rules for the proposed method. The theory is inspired by Bertsekas [4].

Consider a smooth loss function $\mathcal{L}(\theta)$ with $\theta \in \mathbb{R}$ and a constraint $c(\theta)$ expressing the target sparsity through a corresponding bound κ on the L_0-norm of θ. Then, the training objective is given by

$$\min_{\theta} \ \mathcal{L}(\theta) \quad \text{s.t.} \quad c(\theta) = L_0(\theta) - \kappa \leq 0. \tag{6}$$

By introducing a function $\gamma[t]$, with $\gamma[t] = 0$ if $t = 0$ and $\gamma \to \infty$ otherwise, this constrained problem can be expressed as an equivalent unconstrained problem

$$\min_{\theta} \ \mathcal{L}(\theta) + \gamma[(L_0(\theta) - \kappa)^+]. \tag{7}$$

Here $(\cdot)^+$ denotes $\max\{0, \cdot\}$ (ReLU). As the L_0-norm is non-differentiable, we introduce auxiliary variables u of the same shape as parameters θ. This allows us to write Equation (7) as

$$\min_{\theta,u} \ \mathcal{L}(\theta) + \gamma[(L_0(u) - \kappa)^+] \quad \text{s.t.} \quad \theta - u = 0. \tag{8}$$

Applying the augmented Lagrangian method for the newly introduced constrained in Equation (8), yields

$$\min_{\theta,u} \ \mathcal{L}(\theta) + \gamma[(L_0(u) - \kappa)^+] + \lambda^\top(\theta - u) + \frac{1}{2}\mu\|\theta - u\|^2, \tag{9}$$

with Lagrange multipliers λ of the same shape as θ (and u). Note that the latter formulation is equivalent to

$$\min_{\theta} \ \mathcal{L}(\theta) + \rho(\theta, \lambda), \tag{10}$$

with

$$\rho(\theta, \lambda) = \min_{u} \ \gamma[(L_0(u) - \kappa)^+] + \lambda^\top(\theta - u) + \frac{1}{2}\mu\|\theta - u\|^2 \tag{11}$$

$$= \min_{u} \ \lambda^\top(\theta - u) + \frac{1}{2}\mu\|\theta - u\|^2 \quad \text{s.t.} \quad c(u) = L_0(u) - \kappa \leq 0. \tag{12}$$

Given an optimal solution $u^\star$ for Equation (12), we obtain an unconstrained version of the original problem with

$$\rho(\theta, \lambda) = \lambda^\top(\theta - u^\star) + \frac{1}{2}\mu\|\theta - u^\star\|^2, \tag{13}$$

where $u^\star$ is given by

$$u^\star_{\text{unconst.}} = \theta + \frac{\lambda}{\mu}. \tag{14}$$

To meet constraint $c(u)$, there can at most be κ values in u that are non-zero. Let ξ be the number of parameters that need to be zero in order to meet the constraint.

Theorem 1 *The optimal solution of the constrained function $\rho(\theta, \lambda)$ in Equation (12) is given by calculating the optimal unconstrained solution $u^\star_{\text{unconst.}}$ and setting the smallest ξ absolute values in this solution to zero.*

Proof To solve the constrained problem in Equation (12), one defines three sets $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{S}$ that contain indices of elements in u. $\mathcal{P}$ contains all indices of those elements in u that need to be zero in order to meet the constraint $c(u)$. Set $\mathcal{Q}$ contains all indices of elements in u that are not in $\mathcal{P}$. Finally, $\mathcal{S}$ is given as the union $\mathcal{S} = \mathcal{P} \cup \mathcal{Q}$.

It now suffices to show that the optimal values $u_j^\star$ corresponding to indices $j \in \mathcal{Q}$ are equivalent to their unconstrained optimal values given in Equation (14) and that set P contains the indices of the smallest $|u_{i,\text{unconst}}| = |\theta_i + \frac{\lambda_i}{\mu}|$ in u.

The Lagrange function of Equation (12) is

$$L(u) = \rho(u) + \sum_{i \in \mathcal{P}} a_i h_i(u), \tag{15}$$

with Lagrange multipliers a_i[1] and

$$h_i(u) = u_i. \tag{16}$$

The Karush-Kuhn-Tucker (KKT) conditions are then

$$\nabla L(u) = \nabla \rho(u) + \sum_{i \in \mathcal{P}} a_i \nabla h_i(u) \qquad = 0 \tag{17}$$

$$= -\lambda + \mu(u_i - \theta_i) + \sum_{i \in \mathcal{P}} a_i e_i \qquad = 0, \tag{18}$$

with standard basis vectors e_i.
For all indices $j \in \mathcal{Q}$ this directly yields the already known solution for the unconstrained cases [see Equation (14)] of

$$u_j = \theta_j + \frac{\lambda_j}{\mu}, \tag{19}$$

and proofs the former part of the mentioned objective. Indices $i \in \mathcal{P}$ have corresponding values $u_i^\star = 0$ by definition.

Knowing the optimal values $u^\star$ given sets P and $\mathcal{Q}$ leaves the problem of choosing those sets optimally. Comparing the unconstrained solution of ρ to the constrained one immediately solves this. The unconstrained solution yields:

$$\rho_{\text{unconst.}}(u^\star) = \lambda^\top(\theta - u^\star) + \frac{1}{2}\mu\|\theta - u^\star\|^2 \tag{20}$$

$$= \lambda^\top(\theta - \theta - \frac{\lambda}{\mu}) + \frac{1}{2}\mu \left\|\theta - \theta - \frac{\lambda}{\mu}\right\|^2 \tag{21}$$

$$= \sum_i -\frac{\lambda_i^2}{\mu} + \frac{1}{2}\mu\frac{\lambda_i^2}{\mu^2} \tag{22}$$

$$= -\frac{1}{2\mu}\sum_i \lambda_i^2 \tag{23}$$

1. Note that the Lagrange multipliers are denoted by a_i here. This is done to avoid confusion with the Lagrange multipliers λ of the original problem.

The constrained solution with the optimal values $\forall i \in \mathcal{P} : u_i = 0$ and $\forall j \in \mathcal{Q} : u_j = \theta_j + \frac{\lambda_j}{\mu}$ is given by

$$\rho_{\text{const.}}(u^\star) = \lambda^\top(\theta - u^\star) + \frac{1}{2}\mu\|\theta - u^\star\|^2 \tag{24}$$

$$= \sum_{k\in\mathcal{S}} \lambda_k(\theta_k - u_k^\star) + \frac{\mu}{2}(\theta_k - u_k^\star)^2 \tag{25}$$

$$= \sum_{i\in\mathcal{P}} \left(\lambda_i\theta_i + \frac{\mu}{2}\theta_i^2\right) + \sum_{j\in\mathcal{Q}} \left(\frac{-1}{2\mu}\lambda_j^2\right) \tag{26}$$

$$= \sum_{i\in\mathcal{P}} \left(\frac{\mu}{2}\theta_i^2 + \lambda_i\theta_i + \frac{1}{2\mu}\lambda_k^2\right) + \sum_{k\in\mathcal{S}} \left(\frac{-1}{2\mu}\lambda_k^2\right) \tag{27}$$

$$= \frac{\mu}{2}\sum_{i\in\mathcal{P}} \left(u_{i,\text{unconst.}}^{\star 2}\right) + \sum_{k\in\mathcal{S}} \left(\frac{-1}{2\mu}\lambda_k^2\right) \tag{28}$$

$$= \rho_{\text{unconst.}}(u^\star) + \frac{\mu}{2}\sum_{i\in\mathcal{P}} u_{i,\text{unconst.}}^{\star 2}. \tag{29}$$

One can conclude directly that the optimal constrained solution is only dependent on the optimal unconstrained solution and can be selected by choosing the indices of the smallest $||u_i||$ to be in set P, as this minimizes ρ. ■

The optimal elements of u are then given by

$$u_i^\star = \begin{cases} \theta_i + \frac{\lambda_i}{\mu} & \text{if} \quad i \in \mathcal{Q} \\ 0 & \text{else} \quad i \in \mathcal{P}. \end{cases} \tag{30}$$

Finally, this leads to the following unconstrained training objective:

$$\min_\theta \; \mathcal{L}(\theta) + \lambda^\top(\theta - u^\star) + \frac{\mu}{2}\|\theta - u^\star\|^2, \tag{31}$$

and an update rule for λ given by the augmented Lagrangian method:

$$\lambda \leftarrow \lambda + \mu \cdot (\theta - u^\star). \tag{32}$$

Apart from the original loss function, Equation (31) is directly solvable. This yields an additional update rule for parameters θ at every step:

$$\theta \leftarrow \theta - \eta \cdot (\lambda + \mu \cdot (\phi - u^\star)), \tag{33}$$

with a scalar learning rate η, leaving only the known optimization of the loss function $\mathcal{L}(\theta)$ with the update rules defined by Equation (32) and (33).

Appendix B. Implementation

As in [17], we introduce auxiliary parameters ϕ that are initialized to 1 and get pruned instead of the original model parameters θ. ϕ consists of one weight for each group of parameters θ that can be pruned. We study two variants, one for width-pruning and one for depth-pruning. With both approaches, there is negligible parameter overhead, and pruned structures are removed in a smooth way. Further, the complete pruning step can be incorporated into an existing optimizer such that there is no need to change the actual training process otherwise.

B.1. Channel-wise Simultaneous Fine-tuning and Pruning

This subsection describes the usage of the proposed method as a width pruning approach that prunes individual channels in the FFN layers and heads in the attention modules. We refer to this variant as channel-wise simultaneous fine-tuning and pruning (CSFP). As channels in the FFN weights $W_U, W_G \in \mathbb{R}^{d\times k}$ and $W_D \in \mathbb{R}^{k\times d}$ are interconnected, one vector $\phi_{\text{FFN}_i} \in \mathbb{R}^d$ for each FFN layer i is defined. Applying the pruning vector to a weight W_0 yields:

$$W = DW_0, \tag{34}$$

with the diagonal matrix $D = \text{diag}(\phi_{\text{FFN}_i})$ of vector ϕ_{FFN_i}. In this work, vector ϕ_{FFN_i} consistently applies to the up-projection in all FFN layers during training. In case of existing LoRA adapters A and B, this yields:

$$W = D(W_0 + \alpha BA). \tag{35}$$

Similarly, the pruning vector ϕ_{ATTN_i} adapts the value weight matrix W_V in the attention module, as the heads of the $W_Q, W_K, W_V \in \mathbb{R}^{n_{\text{head}}\cdot d_{\text{head}}\times k}$ and $W_O \in \mathbb{R}^{k\times n_{\text{head}}\cdot d_{\text{head}}}$ weights are interconnected as well. Pruning vector $\phi_{\text{ATTN}_i} \in \mathbb{R}^{n_{\text{head}}}$ is to small to directly apply to the weight W_V. The proposed method repeats the values of ϕ_{ATTN_i} corresponding to each head d_{head} times. This creates a new vector ϕ'_{ATTN_i} where each element corresponds to a channel in W_V. The pruning vectors for the FFN layers are pruned together. The same global pruning applies to the pruning vectors in the attention modules. Note that the amount of additional pruning parameters for the attention modules is substantially lower than for the FFN layers. Therefore, they are separately pruned towards the target sparsity to circumvent imbalances in the pruning decisions.

B.2. Module-wise Simultaneous Fine-tuning and Pruning

The depth pruning approach module-wise simultaneous fine-tuning and pruning (MSFP) is described in the following. In the case of MSFP, every FFN layer and attention module gets only one scalar parameter (ϕ_{FFN_i} and ϕ_{ATTN_i}) each. The individual scalar parameters are multiplied by the output of the respective layers. Multiplying the scalar parameters by their respective layer yields the new output h':

$$h'_{\text{FFN}_i/\text{ATTN}_i} = \phi_{\text{FFN}_i/\text{ATTN}_i} \cdot h_{\text{FFN}_i/\text{ATTN}_i}. \tag{36}$$

Again, the individual pruning parameters are aggregated and pruned in a global fashion.

B.3. Further Design Choices

We describe further implementation choices in this part. Setting the pruning parameters ϕ to the value one, at initialization, ensures smooth incorporation into the model, without altering the model output at the start of training.

The pruning parameters ϕ now normally start training in a different order of magnitude than the original parameters θ. Thus, the proposed methods use distinct learning rates $\eta_{\phi,opt}$ and $\eta_{\phi,prune}$ that apply for the used base optimizer $\text{Opt}(\cdot)$ of choice and the parameter update rule defined by Equation (33), respectively. Learning rate schedulers only affect $\eta_{\phi,\text{opt}}$. This is beneficial, as it helps to push against the regularization of the pruning at the start, leading to more exploration of different parameters that could be pruned. At later stages, the influence of the optimizer $\text{Opt}(\cdot)$ on the pruning parameters decreases guided by the scheduler. This helps the pruning algorithm to fixate on the

then chosen parameters and push them to zero.
Once values of ϕ hit zero, clipping them to zero and setting λ to zero at the same time, avoids oscillations in the Lagrangian multipliers after the target is met [9].
Next, the proposed methods update the Lagrange multipliers λ only every n_{lag} steps and treat them as an HP to avoid instabilities due to frequent changes in the choice of currently regularized parameters.

Appendix C. WildGuard Dataset

We use the task of training a guard model on the WildGuard dataset [3, 11]. This dataset consists of human prompts and LLM responses. During training and inference, the model is instructed to evaluate whether a given human prompt and its corresponding LLM response are harmful, respectively. Further, the model is tasked to predict whether the LLM refuses to respond. Like in the original paper, the evaluation criterion is the $F1$-score at inference time [11]. Instead of evaluating single $F1$-scores for the three demanded predictions per sample, all responses are collected and the total $F1$-score over all predictions is calculated. Next to the given split into train and test data, we create a validation set by randomly selecting samples from the training set. The final datasets consist of 80000 training, 6759 validation, and 1725 test samples.

Appendix D. Hyperparameter Optimization

We describe the optimization of the most important HPs of the proposed method in the following. All experiments train the Llama-3.2-1B [2] model on the WildGuard dataset (see Appendix C). In preliminary experiments, we found that the optimal learning rate, independent of the used method (LoRA, BI, LRP, CSFP, or MSFP), is $\eta = 2.5e-4$ for the given task. Similarly, we found that the optimal batch size is $bs = 80$.
The most important HPs of the CSFP and MSFP are the Lagrange multiplier update frequency n_{lag} and the learning rates $\eta_{\phi,\text{opt}}$ and $\eta_{\phi,prune}$ for the auxiliary pruning parameters ϕ (see Appendix B.3). To decrease the amount of HPs to tune, we start by setting $\eta_{\phi,\text{opt}}$ and $\eta_{\phi,\text{prune}}$ to the same value we denote as η_ϕ. We then conducted experiments at a sparsity of $s = 0.25$. Figure 3 and Table 1 shows the $F1$-score on the validation set for the joint HP optimization of n_{lag} and η_ϕ. For both CSFP and MSFP there is an optimum at $n_{\text{lag}} = 200$. Regarding CSFP, the learning rate $\eta_\phi = 0.010$ and for MSFP it is at $\eta_\phi = 0.025$. Those optimal configurations will be used in the following.
To get more insights, we scale the sparsity in the range $s \in \{0.125, 0.250, 0.500, 0.750, 0.875\}$. The results for CSFP and MSFP are shown in Figure 4 and Table 2. Interestingly, the experiments with CSFP show a reasonable scaling with the target sparsity, while the performance of the MSFP variant drops sharply at $s = 0.50$, decaying to $F1 = 0.0$ for higher sparsity levels. This is caused by parameters that are pruned at a late stage in training, where the model is already relying on the then-pruned parameters.

To deal with this, we employ three measures: 1) guided by the literature, we use warmup and cooldown phases of 10% training time each, where no pruning is done, 2) we increase η_ϕ to 0.05 for MSFP as it otherwise struggles to meet the target sparsity. The corresponding experiments are marked with (wc) in Figure 4 and Table 2. While this improves $F1$-scores for high sparsity levels both for MSFP and CSFP, it does not alleviate the sharp performance drop for MSFP after

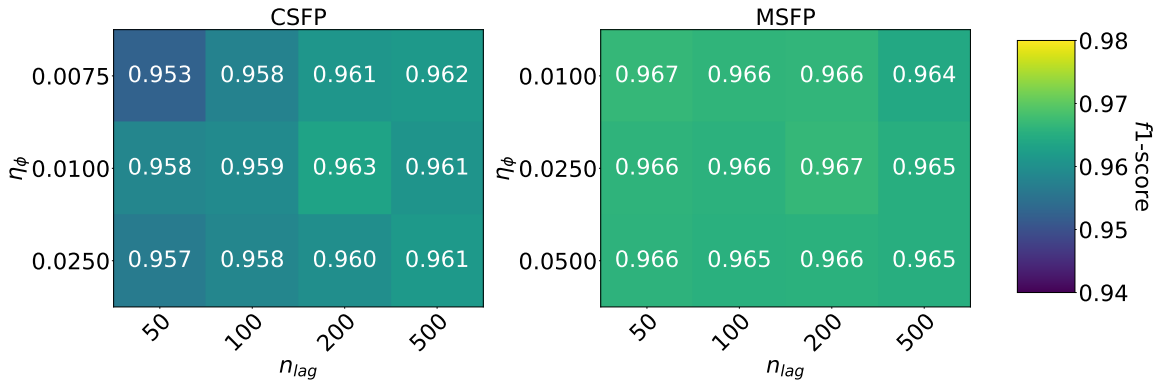

Figure 3: The $f1$-scores on the validation set for different initial learning rates η_ϕ and update frequencies for the Lagrange multipliers n_{lag} for CSFP (left) and MSFP (right).

$s = 0.50$. We conducted three more experiments: 1) separate the values of η_ϕ into $\eta_{\phi,\text{opt}} = 0.02$ and $\eta_{\phi,\text{prune}} = 0.05$ to help prune parameters at an earlier stage in training, 2) apply an early thresholding that directly sets parameters in ϕ to zero, once they are below the threshold of $\text{th}_{\text{early}} = 0.05$, 3) the combination of both configurations above. The corresponding results in Table 2 are marked with (wc, $\eta_{\phi,\text{opt}}$ reduced), (wc, th*) and (wc, th) for 1), 2) and 3), respectively. Only the third option successfully shows reasonable results over all considered sparsity levels. For simplicity, the configurations CSFP (wc) and MSFP (wc, th) are denoted as CSFP and MSFP in the following.

Table 1: The $F1$-scores on the validation set for different initial learning rates $\eta_\phi = \eta_{\phi,\text{opt}} = \eta_{\phi,\text{prune}}$ and update rates for the Lagrange multipliers n_{lag} for CSFP and MSFP.

		n_{lag}			
Method	η_ϕ	50	100	200	500
CSFP	0.0075	0.953	0.958	0.961	0.962
	0.0100	0.958	0.959	0.963	0.961
	0.0250	0.957	0.958	0.960	0.961
MSFP	0.0100	0.967	0.966	0.966	0.964
	0.0250	0.966	0.966	0.967	0.965
	0.0500	0.966	0.965	0.966	0.965

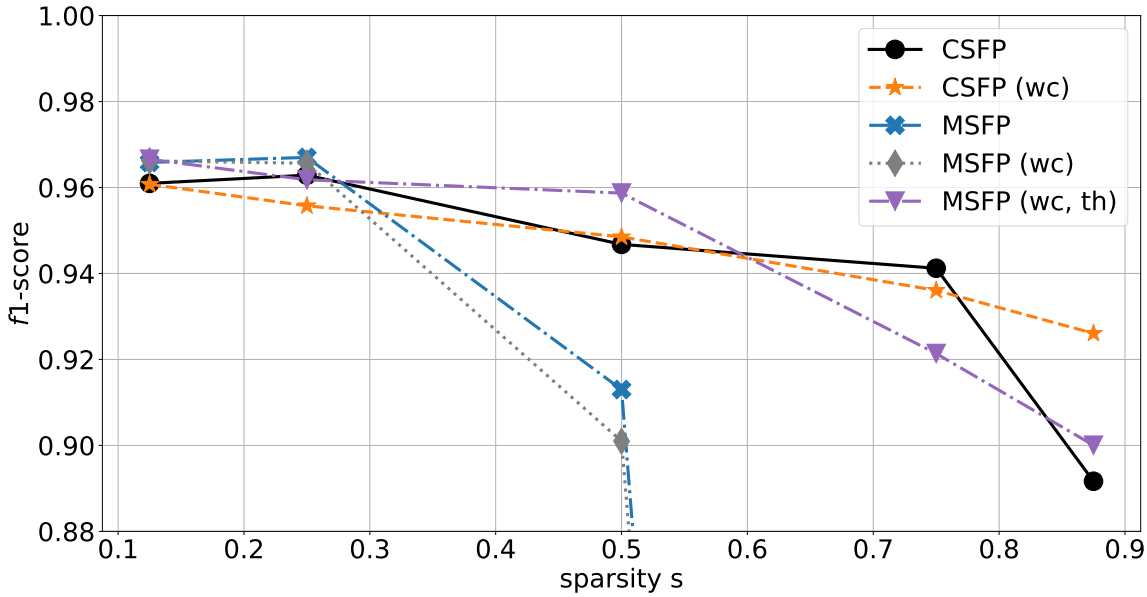

Figure 4: The $F1$-score against the model sparsity on the validation set for CSFP (black) and MSFP (blue). Both for CSFP and MSFP there are results with applied warm-up and cool-down marked by (wc) in the colors orange and green, respectively. Additionally, the red line shows MSFP results with early thresholding and separated learning rates for the pruned parameters.

Table 2: Final $F1$-scores on the validation set at different sparsities and different configurations of CSFP and MSFP.

	s				
Method	0.125	0.250	0.500	0.750	0.875
CSFP	0.961	0.963	0.947	0.941	0.892
CSFP (wc)	0.961	0.956	0.949	0.936	0.926
MSFP	0.966	0.967	0.913	0.000	0.000
MSFP (wc)	0.966	0.966	0.901	0.021	0.391
MSFP (wc, $\eta_{\phi,\text{opt}}$ reduced)	0.964	0.959	0.959	0.554	0.000
MSFP (wc, th*)	0.962	0.966	0.929	0.881	0.750
MSFP (wc, th)	0.967	0.962	0.959	0.921	0.900

Appendix E. Comparison Against Baselines

This section lists the data used in the experiments in Section 4. Table 2 displays the $F1$-scores with standard deviation over two seeds of the fine-tuned and pruned Llama-2-7b-hf model on the WildGuard test set for different methods and sparsity levels. Further, the corresponding inference times over 20 batches are shown. Configurations in the Pareto optimal set are marked in bold.

Table 3: The $F1$-score and inference time t_{inf} of pruned Llama-2-7b-hf models at different sparsities with the respective standard deviation. Values are measured on the test set for the different baseline methods LoRA, BI, LRP and the proposed methods CSFP and MSFP.

Sparsity	Method	$F1$-score	t_{inf}(s)
0.000	LoRA	0.856 ± 0.001	6.073 ± 0.006
0.125	LRP	0.860 ± 0.002	6.622 ± 0.011
	BI	0.858 ± 0.001	5.317 ± 0.005
	CSFP	0.852 ± 0.002	6.129 ± 0.086
	MSFP	0.855 ± 0.001	5.311 ± 0.021
0.250	LRP	0.855 ± 0.003	5.004 ± 0.005
	BI	$\mathbf{0.860 \pm 0.004}$	$\mathbf{4.563 \pm 0.007}$
	CSFP	0.843 ± 0.001	5.328 ± 0.008
	MSFP	0.850 ± 0.001	4.629 ± 0.041
0.500	LRP	0.851 ± 0.000	3.588 ± 0.004
	BI	$\mathbf{0.850 \pm 0.002}$	$\mathbf{3.051 \pm 0.003}$
	CSFP	0.836 ± 0.003	3.861 ± 0.042
	MSFP	$\mathbf{0.852 \pm 0.003}$	$\mathbf{3.222 \pm 0.024}$
0.750	**LRP**	$\mathbf{0.830 \pm 0.001}$	$\mathbf{2.348 \pm 0.001}$
	BI	$\mathbf{0.819 \pm 0.003}$	$\mathbf{1.547 \pm 0.001}$
	CSFP	0.821 ± 0.005	2.181 ± 0.006
	MSFP	$\mathbf{0.827 \pm 0.001}$	$\mathbf{1.803 \pm 0.021}$
0.875	LRP	0.815 ± 0.001	1.726 ± 0.004
	BI	$\mathbf{0.780 \pm 0.002}$	$\mathbf{0.797 \pm 0.001}$
	CSFP	0.809 ± 0.002	1.583 ± 0.003
	MSFP	$\mathbf{0.801 \pm 0.002}$	$\mathbf{1.159 \pm 0.023}$

Appendix F. Complete Hyperparameter Configurations

In this section, we show the full HP configurations used in the different experiments. Table 4 shows the HP configurations dependent on the used model, if not mentioned otherwise in the corresponding experiment. The only differences are for CSFP and MSFP at the beginning of Appendix D, before those phases were introduced.

Table 4: Complete list of hyperparameter configurations in the experiments. Values differ only, if mentioned otherwise in the respective section.

		Model	
HP group	HP	Llama-3.2-1B	Llama-2-7b
General	η (AdamW)	2.5e-4	2.5e-4
	βs (AdamW)	(0.9, 0.999)	(0.9, 0.999)
	η_{min} (Cosine Scheduler)	$\frac{\eta}{100}$	$\frac{\eta}{100}$
	epochs	1	2
	seed	42	$\{1, 42\}$
	batch size	80	80
	accumulation batch size	1	1
	weight decay	0.01	0.01
	LoRA rank	48	48
	LoRA alpha	1	1
	dtype	bfloat16	bfloat16
	max sequence length	1024	1024
	gradient clip norm	0.01	0.01
CSFP	$\eta_{\psi,\text{opt}}$	0.01	0.01
	$\eta_{\psi,\text{prune}}$	0.01	0.01
	n_{lag}	200	200
	warm up, cool down	(10%, 10%)	(10%, 10%)
	μ	1	1
MSFP	$\eta_{\psi,\text{opt}}$	0.02	0.02
	$\eta_{\psi,\text{prune}}$	0.05	0.05
	n_{lag}	200	200
	warm up, cool down	(10%, 10%)	(10%, 10%)
	μ	1	1
LRP	prune frequency	-	10
	warm up, cool down	-	(10%, 10%)
	init sparsity	-	0
BI	calibration dataset	-	full train set
	pruning batch size	-	1