A Proximal Operator for Inducing 2:4-Sparsity

Jonas M. Kübler *kuebj@amazon.com*
Amazon

Yu-Xiang Wang
Amazon and University of California San Diego, USA

Shoham Sabach
Amazon and Technion Israel Institute of Technology, Haifa, Israel

Navid Ansari
Huawei (work done during an internship at Amazon)

Matthäus Kleindessner
Amazon

Kailash Budhathoki
Amazon

Volkan Cevher
Amazon and École Polytechnique Fédéral de Lausanne, Lausanne, Switzerland

George Karypis
Amazon

Reviewed on OpenReview: *https://openreview.net/forum?id=AsFbXRIe4q*

Abstract

Recent hardware advancements in AI Accelerators and GPUs allow to efficiently compute sparse matrix multiplications, especially when 2 out of 4 consecutive weights are set to zero. However, this so-called 2:4 sparsity usually comes at a decreased accuracy of the model. We derive a regularizer that exploits the local correlation of features to find better sparsity masks in trained models. We minimize the regularizer jointly with a local squared loss by deriving the proximal operator for which we show that it has an efficient solution in the 2:4-sparse case. After optimizing the mask, we introduce masked-gradient updates to further minimize the local squared loss. We illustrate our method on toy problems and apply it to pruning entire large language models up to 70B parameters. On models up to 13B we improve over previous state of the art algorithms, whilst on 70B models we match their performance.

1 Introduction

The extensive adoption of large language models has sparked renewed interest in post-training model compression to reduce inference cost and latency (Chitty-Venkata et al., 2023; Park et al., 2024). The most notable techniques are quantization of model weights and activations (Frantar et al., 2023; Dettmers et al., 2022; Frantar et al., 2024) as well as model pruning, i.e., the removal of weights or structures (Frantar & Alistarh, 2023; Sun et al., 2024; Ashkboos et al., 2024). However, pruning entire network structures like columns and rows of weight matrices leads to nonnegligible accuracy drops. Hence, the surge for more flexible sparsity patterns that allow to prune individual weights has sparked. This is accompanied by hardware and software support for efficient sparse matrix multiplications (Pool et al., 2021). In this work we consider structured sparsity, that is we attempt to prune weights of linear models (or linear layers in a neural network)

into a structure where out of each consecutive $M \in \mathbb{N}$ weights $N \in \mathbb{N}$ weights are set to zero. Modern GPUs and AI Accelerators can efficiently represent structured sparsity patterns to compress the memory footprint of the model and accelerate the matrix multiplications. $2:4$-sparsity is the most promising candidate for realizing actual speedups and thus the main focus of this work.

Recent work is generally designed for pruning to unstructured sparsity and has then been applied to structured patterns. In this work, instead, we design a family of regularizers that directly induce structured sparsity. The resulting regularized pruning problems are then solved using the celebrated proximal gradient method. Unlike existing pruning methods, our method induces sparsity *gradually* over the iterations, which can lead to better masks. The proximal gradient method requires solving a *proximal operator* in each iteration, which itself is a nonconvex problem.

The key theoretical contribution of this paper is to show that this non-convex *proximal operator* can be efficiently solved by solving three convex subproblems. We empirically show that these regularizers tend to identify more efficient sparsity masks, reduce the squared loss and can lead to better performance when applied to pruning entire large language models. Our approach natively applies gradient descent on the layer level while finding the mask and after freezing. Our key empirical contribution is that we apply such local gradient descent after masking to previous state-of-the-art methods (Wanda and SparseGPT) and find that we can improve those out of the box. Since Wanda and SparseGPT methods are extensively adopted, we expect that the masked gradient updates will have significant impact.

We defer proofs to Appendix B.

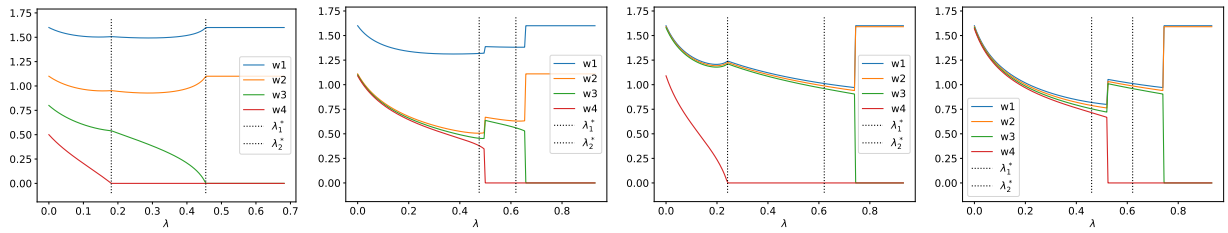

Figure 1: Illustration of the *regularization path*, e.g., optimal solution (w_1, w_2, w_3, w_4) as a function of λ in Equation (7). From left to right, we are showing the result with (a) easy input $z = [1.6, 1.1, 0.8, 0.5]$ (b) nearly-tied $z_{2:4}$ with $z = [1.6, 1.11, 1.1, 1.09]$ (c) nearly-tied $z_{1:3}$ with $z = [1.6, 1.59, 1.58, 1.09]$ (d) All nearly tied with $z = [1.6, 1.59, 1.58, 1.57]$. The two dashed lines in all figures indicates the two critical thresholds λ_1^* and λ_2^* when the 3-sparse solution (and 2-sparse solution) become a *critical point* of Equation (7), thus $\lambda \geq \lambda_1^*$ (or $\lambda \geq \lambda_2^*$) are necessary conditions for the solution to be 3-sparse (or 2-sparse), see Lemma 10 in Appendix B. Observe that in the "easy input" case, as we increase λ, the sparsification is *gradual*, rather than abrupt. In the "hard input" cases, decisions about those that are nearly tied are not made prematurely at smaller λ, the transition to exact structured sparsity happened at a much larger λ.

2 Problem and related work

Large Language Models (LLMs) based on the transformer architecture (Vaswani et al., 2017) have become the workhorse of language modelling. Recent architectures like Llama (Touvron et al., 2023) use decoder only models. Innovations on the attention calculation (Dao, 2023) or its design as grouped-query attention (Ainslie et al., 2023) have in few years already led to notable efficiency improvements. This shifts the focus toward the classic matrix multiplication in linear layers. The idea here is to find an efficient approximation of the weight matrix such that the multiplication can be executed more efficiently as well as that the matrix can be stored efficiently in order to reduce the memory footprint of the matrix multiplication.

2.1 Matrix Compression

Traditional approaches involve low-rank approximations and sparsity. Modern hardware accelerators, which are used to run LLMs, cannot make use of sparsity if the 0s do not follow a regular structure. On the other

hand, if a lot of structure is imposed, for example entire rows or columns are zeroed out, the accuracy drop is often too large.

A method aiming to strike a balance is so-called structured sparsity or N:M sparsity. Here, out of M consecutive weights N are set to zero (Pool et al., 2021). For the case of 2:4 sparsity, one stores only two weights as well as a two-bit index per weight. Thus, for example, if the weights are kept in bfloat16 the memory footprint reduces from $4 \cdot 16 = 64$ bits to $2 \cdot (16 + 2) = 36$ bits, i.e., a compression to 56%. Also since those patterns are efficiently supported in hardware, it can reduce the FLOPs in matrix multiplications by a factor of up to 2x, which is mostly relevant during prompt encoding (aka *prefill*). LLM inference during decoding is on the other hand memory-bound rather than compute bound (Park et al., 2024). There the speedup comes from the memory compression. We note here that this speedup reduces when sparsity is combined with quantization (Jacob et al., 2018; Dettmers et al., 2022; Frantar et al., 2023), as the relative overhead of the position indices grows. With 2:4 sparsity the memory compression and decoding speedup for 16/8/4 bit datatype is 1.77x/1.6x/1.33x.

2.2 Pruning

Many recent works propose new algorithms and heuristics to prune the linear layers in LLMs. Whilst some focus on sparsity (Frantar & Alistarh, 2023; Sun et al., 2024; Dong et al., 2024; Wei et al., 2023) others also remove entire structures, like heads, channels or layers resolving the need for special hardware support (Xia et al., 2024; Ma et al., 2023; Muralidharan et al., 2024). All of these methods have a non-negligible drop in performance metrics, requiring a delicate trade-off between latency/cost and performance degradation of the models. It is known that fine tuning after compression recovers some of the performance drop (Sun et al., 2024; Frantar & Alistarh, 2023; Dong et al., 2024) and recent work have also studied more extensive distillation with a small percentage of pretraining tokens, for example Minitron (Muralidharan et al., 2024) or Llama-3.2.[1]

2.3 One-shot pruning with squared loss

In this work we focus on a local one-shot setting (Frantar & Alistarh, 2023), which we introduce in the following. We consider linear layers in a deep neural network. Let $N, M \in \mathbb{N}$, $N < M$ and $d_i, d_o \in \mathbb{N}$ be the feature input and output dimension, where we assume that d_i is divisible by M. We call a weight matrix $W \in \mathbb{R}^{d_o \times d_i}$ to be N:M sparse if each cell of M consecutive weights contains maximally N non-zero entries $\sum_{i=1}^{M} |W_{r,(k-1)\cdot M+i}|_0 \leq N$ for all $k \in [d/M]$ and for each row $r \in [d_o]$. We will denote the set of N:M sparse matrices as $\mathbb{S}_{N:M}$, leaving the dependency on d_i, d_o implicit.

We assume that the network has been trained to convergence resulting in dense weights W^*. Assuming that we have inputs to the liner layer $X \in \mathbb{R}^{d_i \times n}$ our goal is to find N:M sparse weights that maintain the output of the layer as good as possible in terms of a squared loss (LeCun et al., 1989; Hubara et al., 2021; Frantar & Alistarh, 2023; Sun et al., 2024):

$$\underset{W \in \mathbb{S}_{N:M}}{\arg\min} \frac{1}{n} \|WX - W^*X\|_F^2. \tag{1}$$

This problem can be solved for each row independently, because N:M sparsity imposes the same level of sparsity across all rows and the Frobenius norm decomposes over the rows. Nonetheless, even for a single row, the problem becomes combinatorially hard. Whilst given a sparsity pattern, solving for the optimal weights is a convex problem, the number of masks that need to be searched is $\binom{M}{N}^{d_i/M}$. Since N, M are determined by the hardware and fixed, the complexity grows exponentially with the matrix dimension and in practice we inevitably need to resort to heuristics.

We rewrite the loss in terms of the *Hessian* $H := \frac{XX^\top}{n}$ and using $\|A\|_F^2 = \mathrm{Tr}(AA^\top)$ as

$$\begin{aligned} L(W) :&= \frac{1}{n} \|WX - W^*X\|_F^2 \\ &= \mathrm{Tr}\left((W - W^*)H(W - W^*)^\top\right). \end{aligned} \tag{2}$$

[1] https://huggingface.co/meta-llama/Llama-3.2-1B#training-data

We will also refer to this as *local squared loss* to emphasize that it is on a per-matrix level. Notice that on the diagonal of H we simply have the mean squared activations of the corresponding input channels.

The N:M sparse optimization problem simplifies significantly if the Hessian H is diagonal, meaning that the input features are uncorrelated. In this case the weights cannot compensate for each other, that is each weight is either pruned to zero or kept at its original value. Given a pruning mask $M \in \{0,1\}^{d_o \times d_i}$ the loss is simply

$$\sum_{i,j}(1 - M_{i,j})W^*_{i,j}H_{j,j}W^*_{i,j} =: \sum_{i,j}(1 - M_{i,j})S^2_{i,j}, \tag{3}$$

and we can solve the minimization problem efficiently and optimally by simply pruning those weights with smallest scores $S_{i,j} := |W^*_{i,j}H^{1/2}_{j,j}|$. Although not derived in the way presented here, this is precisely the criterion that Wanda (Weights and Activations) (Sun et al., 2024) proposes for pruning Large Language Models (LLMs) and use it as heuristic even when the Hessian is not diagonal.

Theorem 1 (Wanda is locally optimal for diagonal Hessians)**.** *Let the Hessian H be diagonal, W^* arbitrary and define $M^* \in \{0,1\}^{d_o \times d_i}$ such that for each M cell it has zeros for the N values with smallest score $S_{i,j}$. Then, $M^* \odot W^*$ is a minimizer of problem Equation* (1)*.*

SparseGPT (Frantar & Alistarh, 2023) introduce an iterative heuristic that prunes the weight matrix in column blocks from left to right and takes off-diagonal elements of the Hessian into account. To determine which weights to prune, they use the criterion introduced by Hassibi & Stork (1992) and update the weights in the remaining right blocks, by using the inverse Hessian of the remaining sub-matrix. SparseGPT allows to efficiently prune multiple rows at once, without costly recomputations of the inverse Hessians.

Notice that both SparseGPT and Wanda only require forward passes through the full model to populate the Hessian matrix or mean squared activations for each linear layer, which we dub *one-shot* setting (Frantar & Alistarh, 2023). Both algorithms run in a matter of minutes / hours on a single GPU on large language models up to scales of 100B parameters. We focus on aforementioned *one-shot* setting, where we only allow local updates, but no end-to-end tuning of the entire model. Further recent works one-shot pruning works include DSnoT (Zhang et al., 2024), which provides small improvements in some cases over Wanda and SparseGPT, and ALPS (Meng et al., 2024), which is particularly suited for high sparsity. Dong et al. (2024) investigated other criteria beyond one-shot pruning.

2.3.1 Applicability to arbitrary models with linear layers

While our empirical experiments and investigations focus on large language models, we emphasize that our methodology applies to any pretrained neural network having linear layers. The required information for our algorithm is solely the weight matrices as well as the local Hessians H, which are computed by aggregating input activations over forward passes on calibration data.

2.4 Proximal gradient

The proximal gradient (PG) method is a well-known optimization algorithm designed for solving composite minimization problems, where the objective function can be decomposed into two parts: a smooth differentiable function and a nonsmooth. Such problems arise frequently in many modern applications. In this setting, the objective function typically has the form:

$$\min_w \{f(w) + h(w)\},$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is smooth and differentiable, while $h : \mathbb{R}^n \to (-\infty, \infty]$ is a possibly nonsmooth *structure-inducing* regularization term. To account for this nonsmooth component, the Proximal Gradient (PG) method incorporates a proximal operator alongside the standard gradient step for the smooth part, $f(w)$. The proximal operator, designed to mainly tackle nonsmooth functions, is defined as follows:

$$\text{prox}_h(z) = \arg\min_w \left\{ \frac{1}{2}\|w - z\|^2 + h(w) \right\}.$$

Therefore, to sum-up, starting with an arbitrary w^0, PG method with step-size $t > 0$ generates iteratively a sequence $\{w^k\}_{k\in\mathbb{N}}$ via the following iterative step:

$$w^{k+1} = \text{prox}_{th}(w^k - t\nabla f(w^k)).$$

The PG method is particularly efficient when the smooth part f has a Lipschitz-continuous gradient and when the proximal operator for h has a closed-form solution (e.g., the hard-thresholding operator for ℓ_0-regularized problems). However, if the proximal operator has no closed-form solution it might require an additional tailored optimization algorithm to solve the corresponding problem. Below, we will propose a new function to promote N:M sparsity and discuss in detail its 2:4 proximal operator.

Proximal gradient methods have been extensively studied, particularly in the context of convex optimization problems. For a thorough overview of their theoretical guarantees and extensions, refer to (Beck, 2017) and references therein.

3 Proximal operator for N:M sparsity

Wanda (Sun et al., 2024) and SparseGPT (Frantar & Alistarh, 2023) were both designed primarily for unstructured sparsity and then simply applied to the N:M sparse case by adding these additional constraints. Inspired by PG methods we propose regularizers that are explicitly designed for N:M sparsity and iterate gradient updates on the squared loss (done on the matrix level) with the solution of the proximal operator (which decomposes into the M cells). This leads to a gradual emergence of the pruning mask, where the emerging pattern of one cell can in fact influence the pruning pattern of other cells and avoids committing to a sparsity pattern to early. Wanda and SparseGPT do not have this possibility. Our approach will consume more compute and time, but this is well invested. Compared against the cost of training the models and the inference cost, the additional cost to find a better mask and model is well invested.

3.1 Gradient descent

Each step of PG requires a gradient step on the unregularized loss Equation (2), which has a simple closed-form, see Appendix A:

$$W \leftarrow W - \eta \cdot 2(WH - W^*H), \tag{4}$$

where we can use the largest eigenvalue of the Hessian to set the stepsize $\eta = \frac{1}{2\gamma_{\max}(H)}$, guaranteeing convergence. While each gradient step has complexity $O(d_i^2 d_o)$, it allows us to efficiently use the full parallelization of modern GPUs.

We will use gradient descent during PG, but we also propose it as a method to improve *any* local pruning method. Once the pruning mask M is fixed, we can partially compensate for the pruning loss, by masked gradient updates:

$$W_M \leftarrow W_M - \eta \cdot 2M \odot (W_M H - W^*H). \tag{5}$$

Our first contribution is to show that Wanda and SparseGPT do benefit from such gradient updates after they completed the pruning.

3.2 Regularization and proximal operator

To induce N:M sparsity, we define the following family of regularizers[2]:

$$r_{N:M}(w) := \sum_{\substack{\mathcal{S}\subset[M] \\ |\mathcal{S}|=N+1}} \prod_{j\in\mathcal{S}} |w_j|. \tag{6}$$

Fact 2 (The null space of the regularizer)**.** *For all $w \in \mathbb{R}^M$ we have*

$$r_{N:M}(w) = 0 \text{ if and only if } \|w\|_0 \leq N.$$

[2]In Appendix C we propose a few more, less elaborate, regularizers for 2:4 sparsity.

The corresponding proximal operator is

$$\text{prox}_{\lambda r_{N:M}}(z) := \arg\min_{w \in \mathbb{R}^M} \left\{ \frac{1}{2}\|w - z\|^2 + \lambda r_{N:M}(w) \right\}.$$

The regularizer $r_{N:M}$ promotes N:M sparsity in that N sparse solutions are in the null space of this regularizer. Moreover, as $\lambda \to \infty$, there always exists λ^* (as a function of input z) such that when $\lambda \geq \lambda^*$, the solution of the proximal operator becomes exactly N-sparse (see Figure 1 for an illustration).

For 2:4 sparsity, the proximal operator of interest is

$$\arg\min_{w \in \mathbb{R}^4} \Bigg\{ \frac{1}{2}\|w - z\|^2 + \lambda(|w_1||w_2||w_3| + |w_2||w_3||w_4| + |w_3||w_4||w_1| + |w_4||w_1||w_2|) \Bigg\}. \tag{7}$$

The proximal operator can be used to induce 2:4 sparsity in all settings including: layerwise pruning, finetuning and pretraining using proximal gradient methods or straight-through gradient methods. We focus on layerwise pruning via the squared loss Equation (2). The combined loss function for the 2:4 sparse case is then

$$L_\lambda(W) := L(W) + \lambda \sum_{w \in W} r_{2:4}(w), \tag{8}$$

where $w \in W$ runs over all 2:4 cells of W. Following Fact 2 the matrix W is 2:4 sparse if and only if the regularizer is zero. Importantly the proximal operator corresponding to the regularizer of multiple cells decomposes into multiple 2:4 proximal operators and hence the complexity of solving the proximal operator at each iteration of PG only grows linearly with the dimensionality of the matrix.

3.3 Solution of the 2:4-proximal operator

Solving the 2:4 proximal operator problem is tricky. It is non-convex, non-smooth and appears to require an exhaustive search-style approach. Nonetheless, we have identified interesting mathematical structure of this problem which led to a discovery of very efficient solutions to this problem with provable guarantees.

We first prove that we can always reduce the proximal operator to the case where z is non-negative and sorted $z_1 \geq z_2 \geq z_3 \geq z_4 \geq 0$.

Lemma 3. *To solve problem Equation* (7)*, it suffices to solve*

$$\arg\min_{w \in \mathbb{R}^4_+} f(w), \quad \textit{with} \tag{9}$$

$$f(w) := \Big\{ \frac{1}{2}\|w - z\|^2 + \lambda(w_1 w_2 w_3 + w_2 w_3 w_4 + w_3 w_4 w_1 + w_4 w_1 w_2) \Big\}, \tag{10}$$

where $z_1 \geq z_2 \geq z_3 \geq z_4 \geq 0$*. Moreover, any optimal solution* w^* *of problem Equation* (9) *satisfies that* $w_1^* \geq w_2^* \geq w_3^* \geq w_4^* \geq 0$*.*

Observe that $f(w)$ is smoothly differentiable, but now we have a constrained optimization problem. The same result can generally be stated for N:M-sparsity.

Let us define the restricted version of problem Equation (9) when w_4 is set to 0.

$$\arg\min_{w \in \mathbb{R}^3_+} g(w), \quad \text{with} \tag{11}$$

$$g(w) := \frac{1}{2}\sum_{i=1}^{3}(w_i - z_i)^2 + \lambda w_1 w_2 w_3. \tag{12}$$

As a side remark, the proximal operator for $r_{2:3}$ can be reduced to solving Equation (11).

Lemma 4 (Optimality conditions)**.** *Assume $z_1 \geq z_2 \geq z_3 \geq z_4 > 0$. The solution w^* is one of the following three cases.*

- *2-sparse: in which case $w^* = [z_1, z_2, 0, 0]$.*
- *3-sparse: in which case $w_4^* = 0$ and $w_{1:3}^*$ satisfies*

$$\begin{aligned} 0 < w_1^* &= z_1 - \lambda w_2^* w_3^*, \\ 0 < w_2^* &= z_2 - \lambda w_1^* w_3^*, \\ 0 < w_3^* &= z_3 - \lambda w_1^* w_2^*. \end{aligned}$$

- *Dense: in which case $w^* = [w_1^*, w_2^*, w_3^*, w_4^*] > 0$ satisfies that*

$$w_i^* = z_i - \lambda \cdot \sum_{\{j_1, j_2\} \subset \{j \in [4], j \neq i\}} w_{j_1}^* w_{j_2}^*.$$

In Appendix B.4 we also include necessary conditions on λ for the solutions to be 2-/3-sparse. Whilst the 2-sparse solution is trivial, optimizing for the 3-sparse and dense solution requires more care. We next turn to the Hessians of f and g, see Equation (23) and Equation (24) for their explicit forms.

Lemma 5 (Properties of the Hessians)**.**

- *The set $\mathcal{C}_3 := \{w \in \mathbb{R}^3 \mid \nabla^2 g(w) \succeq 0\}$ is convex.*
- *The set $\mathcal{C}_4 := \{w \in \mathbb{R}^4 \mid \nabla^2 f(w) \succeq 0\}$ is convex.*

This property of the Hessian is crucial, as it allows us to focus on finding a single minimum.

Corollary 6 (No spurious local minima.)**.** *The set of all local minima of $f(w)$ is convex. Moreover, if there exists $w^* \in \arg\min_{w \in \mathbb{R}_+^4} f(w)$ such that $w_i^* > 0$ for all $i \in [4]$ (i.e., in the "Dense" case from Lemma 4), then all local minima of $f(w)$ are global minima.*

The set of all local minima of $g(w)$ is convex. Moreover, if there exists $w^* \in \arg\min_{w \in \mathbb{R}_+^3} g(w)$ such that $w_i^* > 0$ for all $i \in [3]$, then all local minima of $g(w)$ are global minima.

To solve the proximal operator we can thus solve the following constrained optimization problems

$$\min \left\{ g(w) \;\middle|\; w \in \mathbb{R}^3, w \geq 0, \nabla^2 g(w) \succeq 0 \right\}, \tag{13}$$

$$\min \left\{ f(w) \;\middle|\; w \in \mathbb{R}^4, w \geq 0, \nabla^2 f(w) \succeq 0 \right\}. \tag{14}$$

The optimal solutions to Equation (13) and Equation (14) are local minima to Equation (11) and Equation (9) respectively if the solutions have gradient $= 0$. In general, the set of local minima for a non-convex problem can be scattered all over the place, but with Corollary 6 we have shown that for these special functions f and g, the local minima (if they exist at all) form a convex set and have the same objective values.

Problems Equation (13) and Equation (14) are convex optimization problems due to Lemma 5. They can be solved using interior point methods with self-concordant barrier functions. As barrier functions, we can use $-\log\det \nabla^2 f(w)$ and $-\log\det \nabla^2 g(w)$ respectively, as well as $-\log(w_i)$ to encode the positivity constraints (Boyd & Vandenberghe, 2004, Chapter 11.2).

We have now reduced our initial non-convex problem to three convex subproblems, out of which one has a trivial solution. We can then solve those three problems and select the minimizer among the three cases.[3]

Theorem 7. *Algorithm 1 returns an optimal solution to Equation* (9)*.*

[3]If $w^{(4)}$ in Algorithm 2 is a local minimum of f, by Corollary 6 this suffices to know it is the global optimum and there is no need to compute $w^{(2)}, w^{(3)}$. For ease of notation and to parallelize when solving multiple cells, we nonetheless always compute the three cases.

Algorithm 1 Solve Prox with decreasing non-negative input

Require: $z_1 \geq z_2 \geq z_3 \geq z_4 \geq 0$

procedure ProxEnumerate(z, λ)
  $w^{(2)} \leftarrow [z_1, z_2, 0, 0]$
  $w^{(3)} \leftarrow$ [solution to *Equation* (13), 0].
  $w^{(4)} \leftarrow$ solution to Equation (14).
  return $\arg\min_{w \in \{w^{(2)}, w^{(3)}, w^{(4)}\}} f(w)$
end procedure

3.4 Generalization to N:M sparsity.

2:4 sparsity is currently the most relevant sparsity pattern. It is plausible that general N:M sparsity patterns also benefit from a more gradual pruning algorithm. For 1:M sparsity, the proximal operator becomes very simple as it is just a quadratic function and it can be solved in closed form. However, for more general patterns when $N > 2$ solving the proximal operator becomes more challenging. Lemma 4 still holds and it is enough to consider the non-negative sorted case. However, finding the minimizers is harder. In particular it is not obvious how one could replace Lemma 5, i.e., the convexity of the space where the Hessian is convex. There we used that the Hessian in the 2:4 sparse case is linear in w, which will no longer hold for $N > 3$. There might be different arguments to show that those cases can be handled efficiently as well.

3.5 Parallelized Prox by gradient descent

We showed that Equations (13) and (14) are convex optimization problems that can be solved with guarantees for example with interior point methods with self-concordant barrier functions. However, in order to apply this algorithm at scale we need to optimize it further. For example when pruning the MLP down projection of a Llama 70B model (Dubey et al., 2024), the matrix has dimensions 28672 and 8192, meaning that it has around 59 million cells of four weights. Thus at each iteration of Proximal Gradient, we have to solve Algorithm 1 59 million times, which is infeasible if not done in a parallelized fashion using modern GPUs.

In all our experiments, we will thus resort to a gradient descent solver of which we conjecture that it always correctly solves the problem. We will focus on the case of Equation (14), and the case for Equation (13) follows similarly. The constraints $w \geq 0$ is easily enforced by projected gradient descent. The second constraint $\nabla^2 f(w) \succeq 0$ is more computationally expensive and we do not want to compute it at every iteration. Luckily, empirically we find that we actually do not need to enforce this constraint at all. First note two facts

Fact 8. *a)* $w = [0, 0, 0, 0]$ *is always a feasible point of Equation* (14)*. b) Let* γ_{max} *denote the largest eigenvalue of a matrix. For all* $w \in \mathbb{R}^4_+$ *we have* $\nabla^2 f(w) \succeq 0 \Rightarrow \gamma_{max}\left[\nabla^2 f(w)\right] \leq 4$.

Thus, whilst we are in the convex region of the loss, the gradients are Lipschitz with constant 4. Using 1/4 as step size, we are guaranteed to never cross a local minimum as long as we stay in the convex region. In our extensive evaluations we observe that indeed gradient descent does never exit the region of convexity *if the global minimum is dense*. This implies that we can run gradient descent without the need of a barrier function. Whilst we could not find any counterexamples we have not been able to formally prove this and hence state is explicitly as a conjecture.

Conjecture 9. *Let* w^* *be the minimizer of Equation* (14)*, and suppose that* $w^*_i > 0$ *for all* i*. Consider the iterative scheme*

$$w^0 = 0, \quad w^{k+1} = \left(w^k - \eta^k \nabla f(w^k)\right)_+,$$

where the projection $(\cdot)_+$ *is applied elementwise and denotes projection onto* $\mathbb{R}^4_+$*, and* $\eta^k = \frac{1}{4}$*. Then the sequence* $\{w^k\}$ *converges to* w^**.*

Furthermore, each iterate w^k *lies in the feasible set of Equation* (14)*.*

The same considerations and empirical insights apply to the 3-sparse case. Hence, in practice when computing $w^{(3)}$ and $w^{(4)}$ in Algorithm 1 we run the gradient descent based optimization without enforcing to remain

Algorithm 2 Matrix Prox Pruner 2:4

procedure PruneProx($W^*, H, \{\lambda_k\}_k$)
  $W \leftarrow W^*$, $\eta \leftarrow \frac{1}{2\gamma_{\max}(H)}$, $k \leftarrow 0$
  while $W \notin \mathcal{S}_{2:4}$ **do**
    $W \leftarrow W - \eta \cdot 2(HW - HW^*)$
    $W, s, p \leftarrow$ `PosSort`(W)
    $W \leftarrow$ `ProxEnumerate`(W, λ_k)
    $W \leftarrow$ `InvPosSort`(W, s, p)
    $k \leftarrow k + 1$
  end while
  return W
end procedure

in the region of convexity. As soon as we witness that we moved out of the convex region, we can stop the optimization as by Conjecture 9 and Corollary 6 the respective case can not be the optimal one. In practice, we stop when witnessing that the gradient norm increases, as this cannot happen whilst being in the convex region with our choice of step size. Therefore, we slightly deviate from Algorithm 1 in that in line 3 and 4 we do not compute the exact minima if we witness its not the minimum of the three cases anyways. If Conjecture 9 is true, then Theorem 7 holds even with this modification.

In our implementation, the GD-based Algorithm 1 is more than 10x faster than our implementation of the interior point method-based Algorithm 1, and they always obtain numerically the same solution.

3.6 Full algorithm

We can now put together the full matrix pruning algorithm, see Algorithm 2. We define two functions: `PosSort`(W) that returns the weights such that each 2:4 is sorted and without sign as well as s, p that indicate the sign and original position of the weights. `InvPosSort`(W, s, p) which undoes above operation. Furthermore, we use a schedule $\{\lambda_k\}_k$ for the regularization such that $\lambda_k \to \infty$. In this work we consider exponential schedules $\lambda_k = \lambda_0 \beta^k$, for $\lambda_0 > 0$ and $\beta > 1$. Having an exponential schedule guarantees that the regularizer eventually is large enough such that the solution to the proximal operator will be exactly 2:4 sparse, c.f. Section 3.2, which we need to ensure finite runtime of Algorithm 2. In other words: without making assumptions about the problem space, the regularizer eventually needs to be scheduled towards infinity.

4 Experiments

For our experiments unless otherwise stated we use $\lambda_k = \lambda_0 \beta^k$ with $\lambda_0 = 0.01$ and $\beta = 1.01$. Furthermore, we always use the solver Algorithm 2 relying on Conjecture 9. After ending Proximal Gradient, we do 1000 steps of masked gradients according to equation 5 to minimize the local squared loss. Furthermore, we initialize all methods *Wanda* style by transforming $W^*_{i,j} \mapsto W^*_{i,j} H_{j,j}^{1/2}$, $H_{i,j} \mapsto H_{i,j} H_{j,j}^{-1/2} H_{i,i}^{-1/2}$. This helps the proximal pruning to not commit to weights that seem important solely in terms of magnitude, but takes the respective mean squared activations into account (Sun et al., 2019). After pruning this transformation is reversed. All experiments can run on a single NVIDIA A100 GPU with 40GB of memory, but we used multiple GPUs in parallel to speed up the experimentation.

4.1 Toy experiments

We start with a simple experiment to illustrate the inner workings of our algorithm and the shortcoming of Wanda and SparseGPT. We consider the 2:4 sparse case of a single row with dense weights $w^* = (0, 5, 3, 2,\ 0, 5, 5, 2)$. We further assume that the Hessian has ones on the diagonal and zero elsewhere except that the fourth and eighth weight are perfectly correlated (value 1 on the respective off-diagonal elements). In this case the optimal 2:4 sparse solution is $w = (0, 5, 0, 4,\ 0, 5, 5, 0)$. The difficulty here is that the fourth

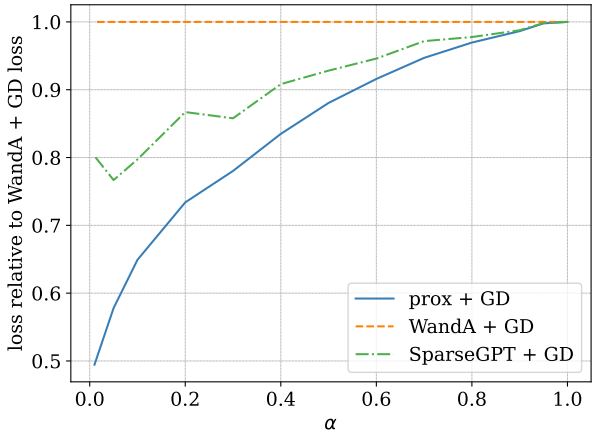

Figure 2: Toy experiment with generated weights and Hessians. Without correlation ($\alpha = 1$) all considered pruning methods result in the same mask and loss. As we increase the correlations, our proximal approach leads to the best mask as shown by the smallest loss.

and eighth weight if looked at individually within their 2:4 cell, would be pruned, but instead we can merge them into one weight and prune another weight incurring a smaller loss.

Since Wanda (Sun et al., 2024) completely ignores the correlations, it prunes to $w = (0, 5, 3, 0,\ 0, 5, 5, 0)$, which is suboptimal. SparseGPT (Frantar & Alistarh, 2023) would in a first step prune the first $'2'$ because it can completely absorb it into the last weight due to their correlation $w \mapsto w = (0, 5, 3, 0,\ 0, 5, 5, 4)$. However then in the second step, when pruning the second cell, it drops the combined $'4'$ because it incurs lower loss than pruning one of the $'5'$s $w \mapsto w = (0, 5, 3, 0,\ 0, 5, 5, 0)$. Therefore, the SparseGPT heuristic can lead to deferred loss.

Our proximal optimizer in turn adapts to the situation. In the second cell, the $'2'$ is quickly pushed to 0. This in turn increases the gradient magnitude of the first $'2'$, pushing it eventually over the value of the $'3'$ in the first cell. To simplify the understanding, we animated the evolution of the weights with a gif → LINK.

Next we consider the effect of correlation between input features more structurally. Therefore we generate synthetic Hessians and random weights:

```
d=1024
diag = torch.diag(torch.rand(d))
Z = torch.randn(d, d) / float(d)**(1/2)
Z = Z @ Z.T
hessian = alpha * diag + (1 - alpha) * Z
weights = torch.randn(1, d)
```

We then prune with all the considered methods and after masking optimize the weights with gradient descent. Results are discussed and shown in Figure 2.

4.2 Scheduling λ

During the course of proximal gradient, we increase the regularizer λ with an exponential schedule to guarantee, that no matter what input, eventually the weights will be 2:4 sparse. In this work we consider exponential schedules $\lambda_k = \lambda_0 \beta^k$, for $\lambda_0 > 0$ and $\beta > 1$. Clearly, good choices of those hyperparameters depend on each problem and are expensive to ablate when done for each problem separately. We show the impact of different choices on the local loss and the runtime in the Appendix Figure 3. Notice that the optimization problem equation 9 is not scale invariant when scaling z alone. However, if w^* is the optimal solution to the problem z, λ, then cw^* is the optimal solution for the problem $(cz, \lambda/c)$. Since it is too

Table 1: Validation Perplexity of OpenLlama (3B/7B/13B) and Llama3.1 (8B/70B) (Pretrained/Instruct). . All the methods with *+GD* are an innovation of the present work.

	3b_v2		7b_v2		13b		8B		8B Inst		70B		70B Inst	
Method	C4	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4	Wiki	C4	Wiki
dense	9.68	14.15	8.84	12.15	8.11	11.52	9.68	7.93	11.28	7.27	7.32	3.34	8.26	3.71
wanda	29.48	60.32	17.63	27.02	13.17	35.45	36.32	27.26	43.30	26.67	13.42	<u>10.25</u>	13.68	<u>8.74</u>
wanda+GD	18.23	33.05	14.35	21.85	11.73	18.17	22.19	20.46	28.45	20.71	12.32	10.38	13.08	8.91
sp.gpt	18.76	33.78	14.24	22.29	11.91	19.51	24.95	23.18	35.55	26.19	13.00	10.93	14.05	9.37
sp.gpt+GD	16.72	29.58	13.31	20.83	11.43	<u>17.19</u>	21.00	19.86	27.54	20.85	<u>12.06</u>	10.49	<u>13.06</u>	8.91
DSnoT	26.54	44.17	17.62	26.17	13.71	32.89	40.66	30.40	48.80	31.46	14.34	10.54	14.51	9.16
prox+GD	<u>16.27</u>	<u>28.59</u>	<u>13.19</u>	<u>20.55</u>	<u>11.40</u>	17.71	<u>20.86</u>	<u>19.83</u>	<u>26.46</u>	<u>20.20</u>	12.33	10.38	13.12	8.91

expensive to extensively search the hyperparameters for each problem individually, we propose to consider

$$\lambda_k = \frac{\tilde{\lambda}_0}{\text{Mean}[|W^*|]}\beta^k. \tag{15}$$

4.3 Pruning large language models

To prune LLMs, we use 2 million tokens from the c4 dataset (Raffel et al., 2020), which we pack with end-of-sequence tokens and chunk into 1024 sequences[4] of lengths 2048. We estimate all Hessians from the unpruned model, and do not propagate the pruning errors as we find that this has little impact (Appendix D.1). As baselines we run Wanda (Sun et al., 2024) and SparseGPT (Frantar & Alistarh, 2023) both with their original algorithms as well as with our innovative additional 1000 gradient descent steps after masking, see Section 3.1, where we observed that the local loss has usually converged after 1000 steps as shown in Appendix Figure 5. We also run DSnoT (Wanda) (Zhang et al., 2024). For the proximal, for small models we used our default setting $\lambda_k = \lambda_0\beta^k$ with $\lambda_0 = 0.01$ and $\beta = 1.01$. For the 70B models, we observed that this would require more than 2000 iterations of PQ. We thus used the heuristic from Equation (15) and the results of Figure 3 and selected $\beta = 1.005$, $\tilde{\lambda}_0 = 1 \cdot 10^{-3}$ to strike a good balance between performance and runtime. Since, `prox` already uses gradient updates during pruning, we recommend to always use it together with gradient updates after freezing the mask. For completeness though, in Appendix D.3 we also include a small ablation on the effect of masked gradient steps for `prox`.

We prune models from the openLlama family (Geng & Liu, 2023) and the Llama-3.1 models (Dubey et al., 2024), see Table 1. We evaluate the models both on in-distribution validation data from c4, as well as out of distribution data from WikiText2 (Merity et al., 2016).

On models up to 13B parameters, we see that the proximal approach consistently outperforms the other methods at least in-distribution (C4) and mostly also on wikitext data (wiki). In the appendix Figure 4 we show that this is related to an improvement in local squared loss. Furthermore, Wanda and SparseGPT both benefit clearly from our proposed masked gradient descent based updates after pruning with the respective strategy.

Interestingly, whilst `prox` tends to give the best perplexity, it turns out that our innovation on using masked gradient updates (Section 3.1) is empirically more relevant, as it consistently improves the perplexity on C4. As a by-product of our work we thus identified a method that can be plugged on top of existing SOTA one-shot pruning methods. In Table 3 we evaluate the pruned Llama-3.1 8B Instruct model on 6 Downstream tasks and find that on average we improve the SOTA more than 3%.

On the 70B models, whilst the gradient updates continue to improve the in-distribution loss, it can harm the wiki perplexity in case of Wanda, which can indicate a slight overfitting to the domain of the calibration

[4]We emphasize that all methods in our experiments use exactly the same number of calibration samples. In Appendix D.2 we further ablate the effect using different number of calibration samples.

data. Furthermore, on the 70B models we see that the method to find the mask has a very minor impact on the final perplexity. On the 70B model scale, we thus recommend to use wanda for pruning instead of SparseGPT or prox and rather invest resources into having a more balanced and larger calibration sample to estimate the Hessian from in order to mitigate domain speciliazation.

We can qualitatively define a hierarchy between the investigated methods by how much information of the Hessian matrix is used whilst identifying the mask. Wanda uses only the diagonal entries (see also Theorem 1), SparseGPT uses already the full Hessian, and prox makes even heavier use of the Hessian (see our toy experiment of Figure 2). We observe that on the smaller models, SparseGPT does quite consistently outperform Wanda, hence it seems that the off-diagonal elements of the Hessian are relevant. It is those cases, where doubling-down on using Hessian information (through prox or masked gradient updates) further improves performance. On the other hand, on the 70B scale using plain Wanda seems close to optimal – neither SparseGPT, prox, nor masked gradient updates improve there relevantly. This seems to indicate that the behavior on this scale is largely determined by the diagonal entries of the Hessian matrix – in which case wanda (by our Theorem 1) is optimal. However, the difficulty in analyzing and predicting the end-to-end behavior from the properties of the matrices is that the weight matrix and the Hessian need to be considered jointly. Existing methods to characterize matrices like Kruskal Rank (Kruskal & Wallis, 1952), Mutual Coherence, or the restricted isometry property usually consider a single matrix in isolation.

Overall, we find that a 2:4 sparse model is consistently worse than the dense smaller model of the same class which is consistent with the results on Llama 1 and 2 models found by Sun et al. (2024, Table 3). A potential way to mitigate this large drops is to only partially prune the model, e.g., only certain modules or only a fraction of the model's layers. This can be considered for all pruning methods equally and is an orthogonal direction for future work. Additionally, pruned models can be further fine-tuned to specific downstream tasks, which can further mitigate the loss in quality.

5 Discussion

We have introduced a method to gradually induce structured sparsity in pretrained linear layers, for example of large language models. The immediate objective is to minimize the linear squared loss at the matrix level. We showed that a local improvement of the squared loss leads to an improvement in model perplexity. On the theoretical side, we showed how to efficiently solve the complex proximal operator for 2:4 sparsity.

On the empirical side, we observe that the sparsity mask does not make a huge difference for recent LLMs, but that our introduced local gradient based updates can also be used on top of existing local pruning methods and improve their performance. In particular, our proposed masked-gradient updates can easily be used to improve the widely adopted methods Wanda and SparseGPT. We believe that the masked gradient updates will be widely adopted.

Whilst we thus improved the state of the art on one-shot pruning, the resulting pruned models clearly are not good enough to be of direct practical relevance. Looking at Table 1, we see that the pruned 13B/7B models fall clearly behind the dense 7B/3B model. However, for a 2x matrix compression this should be the lowest bar a model has to take in order to be useful. This is even more because whilst the structured sparsity reduces the latency and footprint of matrix multiplications, it does not decrease the sizes of in-/output activations or the KV-cache.

One benefit of the proximal approach and the theoretical insight we provided is that they can also be applied during pretraining together with a regular pretraining objective.[5] Other pruning approaches discretely interrupt the optimization, whilst the proximal operator is applied at every step, hence gradually pushing the model to sparsity. This could pave the road for more powerful sparse models with less quality degradation.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of*

[5]Concurrent work Fang et al. (2024) found that end-to-end learning of masks is beneficial also post training.

the 2023 Conference on Empirical Methods in Natural Language Processing, 2023.

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 *The Twelfth International Conference on Learning Representations*, 2024.

Amir Beck. *First-order methods in optimization*, volume 25. SIAM, 2017.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Krishna Teja Chitty-Venkata, Sparsh Mittal, Murali Emani, Venkatram Vishwanath, and Arun K Somani. A survey of techniques for optimizing transformer inference. *Journal of Systems Architecture*, pp. 102990, 2023.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv:2307.08691*, 2023.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems*, 2022.

Peijie Dong, Lujun Li, Zhenheng Tang, Xiang Liu, Xinglin Pan, Qiang Wang, and Xiaowen Chu. Pruner-zero: Evolving symbolic pruning metric from scratch for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

Abhinav Dutta, Sanjeev Krishnan, Nipun Kwatra, and Ramachandran Ramjee. Accuracy is not all you need. *arXiv preprint arXiv:2407.09141*, 2024.

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. Maskllm: Learnable semi-structured sparsity for large language models. *NeurIPS*, 2024.

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

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *ICLR*, 2023.

Elias Frantar, Roberto L Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. Marlin: Mixed-precision auto-regressive parallel inference on large language models. *arXiv:2408.11743*, 2024.

Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, May 2023. URL `https://github.com/openlm-research/open_llama`.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 1992.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Joseph Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. *Advances in neural information processing systems*, 2021.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

William H Kruskal and W Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association*, 47(260):583–621, 1952.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 1989.

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

Xiang Meng, Kayhan Behdin, Haoyue Wang, and Rahul Mazumder. ALPS: Improved optimization for highly sparse one-shot pruning for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

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

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. *arXiv:2407.14679*, 2024.

Youngsuk Park, Kailash Budhathoki, Liangfu Chen, Jonas M. Kübler, Jiaji Huang, Matthäus Kleindessner, Jun Huan, Volkan Cevher, Yida Wang, and George Karypis. Inference optimization of foundation models on ai accelerators. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024.

Jeff Pool, Abhishek Sawarkar, and Jay Rodge. Accelerating inference with sparsity using the nvidia ampere architecture and nvidia tensorrt, 2021.

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. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *ICLR*, 2024.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression, 2019.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *EMNLP*, 2023.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations*, 2024.

Yuxin Zhang, Lirui Zhao, Mingbao Lin, Sun Yunyun, Yiwu Yao, Xingjia Han, Jared Tanner, Shiwei Liu, and Rongrong Ji. Dynamic sparse no training: Training-free fine-tuning for sparse LLMs. In *The Twelfth International Conference on Learning Representations*, 2024.

A Gradient and Hessian

We have the following loss function Equation (2):

$$L(W) = \mathrm{Tr}\left((W - W^*)H(W - W^*)^\top\right).$$

The loss is simply the sum of the loss of each individual row and we e can minimize this on a per-row basis ($w \in \mathbb{R}^{d_i}$)

$$L(w) = (w - w^*)^\top H(w - w^*). \tag{16}$$

From here we can simply compute the gradients:

$$\nabla L(w) = 2H(w - w^*) = 2Hw - 2Hw^*. \tag{17}$$

We can the stack this again for the whole matrix and obtain:

$$\nabla L(W) = 2WH - 2W^*H. \tag{18}$$

Notice that the second term can be precomputed once, and does not need to be computed again at each iteration. Computing the second order derivatives per row we find $\nabla^2 L(w) = 2H$, hence also our naming conventions (ignoring the factor of 2).

B Proofs

B.1 Proof of Theorem 1

Proof. We first show that a diagonal Hessian implies that for any mask, the optimal values of the remaining weights match their dense counterparts. Therefore, let M be an arbitrary binary mask and assume that the Hessian is diagonal, i.e., $H_{ij} = 0$ whenever $i \neq j$. We use the formulation of Equation (2) where we explicitly add the mask

$$\begin{aligned} L(W, M) &= \mathrm{Tr}\left((M \odot W - W^*)H(M \odot W - W^*)^\top\right) && (19)\\ &= \sum_{i,j}(M_{i,j}W_{i,j} - W^*_{i,j})^2 H_{j,j} && (20)\\ &= \sum_{i,j|M_{i,j}=1}(W_{i,j} - W^*_{i,j})^2 H_{j,j} + \sum_{i,j|M_{i,j}=0}(W^*_{i,j})^2 H_{j,j}. && (21) \end{aligned}$$

Notice that the second term solely depends on the mask and not on the weights of W. Furthermore, the first term can be set to zero by matching the original weights. By definition WandA selects a mask that minimizes the second term whilst keeping the other weights fixed, hence minimizing the overall squared loss. □

B.2 Proof of Fact 2

Proof. The "if" part is trivial. If $\|w\|_0 \leq N$, then for any index subset $\mathcal{S} \subset [M]$ with $|\mathcal{S}| = N + 1$, at least one of the coordinate of $w_{\mathcal{S}}$ is 0. To see the "only if" part, if suffices to prove the contra-positive. If $\|w\|_0 > N$, let $\tilde{\mathcal{S}}$ to be the first $N + 1$ coordinates of w in $\tilde{\mathcal{S}}$, thus $r_{N:M}(w) \geq \prod_{i \in \tilde{\mathcal{S}}} |w_j| > 0$. □

B.3 Proof of Lemma 3

Proof. First observe that the objective function of problem Equation (7) is invariant to joint permutations and signs of w and z. In other words, if $\tilde{w}^*$ is an optimal solution of problem Equation (7) for a given input $\tilde{z} \in \mathbb{R}^4$, then for any permutation Π and any diagonal matrix D with diagonal elements being -1 or 1, the $D\Pi\tilde{w}^*$ is an optimal solution of problem Equation (7) when the input is $D\Pi\tilde{z}$ and having the same optimal value. Therefore, without the loss of generalization, we can choose Π to be the permutation that sorts the input into a descending order by the absolute value of $|z|$, and $D = \mathrm{diag}(\mathrm{sign}(\Pi z))$.

Next, we observe that for a non-negative input z, any optimal solution of problem Equation (7) is always non-negative. To see this, if w_i^* is negative and $z_i > 0$, replacing it with $|w_i^*|$ retains the regularizer but $(|w_i^*| - z_i)^2 < (w_i^* - z_i)^2$. If $z_i = 0$, the corresponding optimal solution is $w_i^* = 0$, which is non-negative. Thus, we can remove the absolute values and focus on solving problem Equation (9).

Moreover, we observe that a sorted (descent order) input z implies that any optimal solution is also sorted in a descent order. Indeed, we take w^* to be an optimal solution with $w_i^* < w_j^*$ for some $1 \leq i < j \leq 4$. Since the regularizer is invariant to permutations, we focus on the quadratic part and show that in this case $\tilde{w}$, which is defined by $\tilde{w}_i = w_j^*$, $\tilde{w}_j = w_i^*$ and $\tilde{w}_k = w_k^*$ for all $k \neq i, j$, we have that

$$\begin{aligned}
\|w^* - z\|^2 - \|\tilde{w} - z\|^2 &= (w_i^* - z_i)^2 + (w_j^* - z_j)^2 - (\tilde{w}_i - z_i)^2 - (\tilde{w}_j - z_j)^2 \\
&= (w_i^* - z_i)^2 + (w_j^* - z_j)^2 - (w_j^* - z_i)^2 - (w_i^* - z_j)^2 \\
&= 2(w_j^* z_i + w_i^* z_j - w_i^* z_i - w_j^* z_j) \\
&= 2(w_j^* - w_i^*)(z_i - z_j) \\
&\geq 0,
\end{aligned}$$

where the last inequality follows from the facts that $w_i^* < w_j^*$ and $z_i \geq z_j$. This shows that $\tilde{w}$, which is a sorted (descent order) vector, is also an optimal solution.

Combining these two observations means that problem Equation (7), which is formulated for any input z, can be rewritten only for non-negative sorted inputs. Note that we can always invoke the above observations to recover a solution to the original problem by first solving problem Equation (9) to find w^* and then $\Pi^{-1} D^{-1} w^*$ is an optimal solution of problem Equation (7). □

B.4 Proof of Lemma 4

We first state a slighlty extended version of Lemma 4

Lemma 10 (Optimality conditions – Extended)**.** *Let the objective function in Equation* (9) *be* $f(w)$*. Assume* $z_1 \geq z_2 \geq z_3 \geq z_4 > 0$*. The solution* w^* *must be in one of the following three cases.*[6]

- 2*-sparse: in which case* $w^* = [z_1, z_2, 0, 0]$*.*
- 3*-sparse: in which case* $w_4^* = 0$ *and* $w_{1:3}^*$ *satisfies that*

$$\begin{aligned}
0 < w_1^* &= z_1 - \lambda w_2^* w_3^*, \\
0 < w_2^* &= z_2 - \lambda w_1^* w_3^*, \\
0 < w_3^* &= z_3 - \lambda w_1^* w_2^*.
\end{aligned}$$

- *Dense: in which case* $w^* = [w_1^*, w_2^*, w_3^*, w_4^*] > 0$ *satisfies that*

$$w_i^* = z_i - \lambda \cdot \sum_{\{j_1, j_2\} \subset \{j \in [4], j \neq i\}} w_{j_1}^* w_{j_2}^*.$$

Moreover if $z_1, z_2 > 0$*,*

1. $\lambda \geq z_3/(z_1 z_2)$ *is a necessary condition for the solution to be in the* 2*-sparse regime.*
2. $\lambda \geq z_4/(w_1^* w_2^* + w_2^* w_3^* + w_1^* w_3^*)$ *is a necessary condition for the solution to be in the* 3*-sparse regime, for* $w_{1:3}^*$ *to be the solution to Equation* (11)*. A weaker necessary condition is* $\lambda \geq z_4/(z_1 z_2 + z_2 z_3 + z_1 z_3)$*.*

[6] If we have equality in some of the inputs, i.e., $z_i = z_j$, permuting those two inputs also leads to valid solutions. We do not give precedence to either of those solutions but simply chose the one that follows the order provided by the sorting algorithm. For the proof we ignore this degeneracy to keep it simple.

Proof. If an optimal solution w^* of problem Equation (9) is 2-sparse, it means that $r_{2:4}(w^*) \equiv 0$. Therefore, w^* is a minimizer of the quadratic term. Moreover, since w^* is sorted in a descent order it means that $w_1^*, w_2^* > 0$ and $w_3^* = w_4^* = 0$. Therefore, the desired result immediately follows.

If an optimal solution w^* of problem Equation (9) is 3-sparse, it means that $w_4^* = 0$. Therefore, w^* is also an optimal solution of problem Equation (11). Moreover, since w^* is a non-negative 3-sparse vector it follows that $w_1^*, w_2^*, w_3^* > 0$. This implies that the non-negative constraint of problem Equation (11) is inactive and therefore an optimal solution must be a stationary point of the objective function. The desired result follows by zeroing the gradient of the objective function of problem Equation (11).

If an optimal solution w^* of problem Equation (9) is dense, it follows that $w^* > 0$. Then, the non-negative constraint of problem Equation (9) is inactive at w^*. Therefore, w^* must be a stationary point of the objective function of problem Equation (9).

Since the optimization problem Equation (9) is a minimization of the (non-convex) differentiable objective function f over the non-negative constraint, it is well known that the corresponding KKT conditions are necessary for critical points w^*, which are compactly given by $\nabla f(w^*) \geq 0$, $w^* \geq 0$ and $\nabla f(w^*)^T w^* = 0$. Therefore, from the third condition it follows that for any positive element of w^* the corresponding partial derivative of f must be zero. Moreover, by writing the gradient of the objective function of problem Equation (9), we get that

$$\nabla f(w^*)^T w^* = (w^* - z + \lambda \nabla r_{2:4}(w^*))^T w^* = \|w^*\|^2 - z^T w^* + 3\lambda r_{2:4}(w^*),$$

where the last equality follows from the fact that $\nabla r_{2:4}(w)^T w = 3r_{2:4}(w)$ for any $w \in \mathbb{R}^4$. From here we immediately see that $z \geq w^*$ and if $r_{2:4}(w^*) \neq 0$ then $\lambda = (z^T w^* - \|w^*\|^2)/3r_{2:4}(w^*)$. From the first condition, that is $\nabla f(w^*) \geq 0$, we get that $\lambda \geq (z_i - w_i^*)/\nabla_i r_{2:4}(w^*)$ for all $1 \leq i \leq 4$ that $\nabla_i r_{2:4}(w^*) \neq 0$.

Since the optimal solutions must be monotonically decreasing in i there are no other cases.

The two other necessary conditions about λ stem from the KKT conditions of a critical point (with active constraints) for the constrained optimization problem. Specifically, the Lagrangian of f $\mathcal{L}(w, \nu) = f(w) - \nu^T w$.

The conditions for critical points (including those at 0) are that there exists $\nu \in \mathbb{R}^4$ such that

1. Stationarity: $\nabla_w \mathcal{L}(w, \nu) = \nabla f(w) - \nu = 0$
2. Complementary slackness: $\nu_i w_i = 0$ for $i = 1, 2, 3, 4$.
3. Primal feasibility: $w \geq 0$
4. Dual feasibility: $\nu \geq 0$.

For any critical point to be 2-sparse i.e., $w_1 > 0, w_2 > 0, w_3 = w_4 = 0$, it needs to satisfy complementary slackness which implies $\nu_1 = \nu_2 = 0$. Moreover, stationarity condition gives $w_1 = z_1, w_2 = z_2$,

$$\nu_3 = \frac{\partial}{\partial_{w_3}} f(w) = w_3 - z_3 + \lambda w_1 w_2 = -z_3 + \lambda z_1 z_2$$

and

$$\nu_4 = \frac{\partial}{\partial_{w_4}} f(w) = -z_4 + \lambda z_1 z_2$$

Now the dual feasiblity condition $\nu_3, \nu_4 \geq 0$ gives the condition that $\lambda \geq z_3/(z_1 z_2)$.

Similarly, for any critical point to be 3-sparse, i.e., $w_1 > 0, w_2 > 0, w_3 > 0, w_4 = 0$, we have $\nu_1 = \nu_2 = \nu_3 = 0$, and that there exists

$$\nu_4 = \frac{\partial}{\partial_{w_4}} f(w) = -z_4 + \lambda(w_1 w_2 + w_2 w_3 + w_3 w_1) \geq 0.$$

w_1, w_2, w_3 can be solved by the following non-linear system of equation

$$\begin{cases} w_1 + \lambda w_2 w_3 = z_1, \\ w_2 + \lambda w_1 w_3 = z_2, \\ w_3 + \lambda w_1 w_2 = z_3. \end{cases} \tag{22}$$

For this reason, if we are able to first identify a solution to Equation (22) then verify that the $\nu_4 \geq 0$, then we certified that this solution is a critical point — a necessary condition for this sparse solution to be a global optimal solution. □

B.5 Proof of Lemma 5

Proof.

$$\nabla^2 g(w) = \begin{bmatrix} 1 & \lambda w_3 & \lambda w_2 \\ \lambda w_3 & 1 & \lambda w_1 \\ \lambda w_2 & \lambda w_1 & 1 \end{bmatrix} \quad \text{and} \tag{23}$$

$$\nabla^2 f(w) = \begin{bmatrix} 1 & \lambda(w_3+w_4) & \lambda(w_2+w_4) & \lambda(w_2+w_3) \\ \lambda(w_3+w_4) & 1 & \lambda(w_1+w_4) & \lambda(w_1+w_3) \\ \lambda(w_2+w_4) & \lambda(w_1+w_4) & 1 & \lambda(w_1+w_2) \\ \lambda(w_2+w_3) & \lambda(w_1+w_3) & \lambda(w_1+w_2) & 1 \end{bmatrix}. \tag{24}$$

Note that the conditions that $\nabla^2 g \succeq 0$ and $\nabla^2 f \succeq 0$ are *Linear Matrix Inequalities*, thus convex. □

B.6 Proof of Corollary 6

Proof. It suffices to check the definition of a convex set, i.e., for any pair of local minima, their convex combination is also a local minimum.

Let $\mathcal{C}_4 := \{w \in \mathbb{R}^4 \mid \nabla^2 f(w) \succeq 0\}$ as in Lemma 5, where we showed that it is convex. Let $u^*, v^* \in \mathbb{R}^4$ be local minima of f. Then, since $u^*, v^* \in \mathcal{C}_4$ and $\mathcal{C}_4$ is convex, for any $0 \leq t \leq 1$, f is convex at $w(t) := tu^* + (1-t)v^*$.

By the definition of $\mathcal{C}_4$ when restricting to this set, f is a convex function. By the first order definition of convex function

$$f(u^*) \geq f(v^*) + (u^* - v^*)^T \nabla f(v^*) = f(v^*)$$

similarly

$$f(v^*) \geq f(u^*) + (v^* - u^*)^T \nabla f(u^*) = f(u^*)$$

thus $f(u^*) = f(v^*)$.

By convexity of f on the line segment, $f(w(t)) \leq tf(u^*) + (1-t)f(v^*) = f(u^*)$ for all t. On the other hand, as $t \to 0$, we also have $f(w(t)) \geq f(u^*)$ due to that u^* is a local minima. These two conditions imply that $f(w(t)) = f(u^*)$ for all $t \in [0,1]$, i.e., $w(t)$ is also a local minimum.

To prove the second part we first show that the minimum of f over $\mathbb{R}^4_+$ exists. Therefore note that for any $w \in \mathbb{R}^4_+$ if $w_i > z_i$, we have $\partial_i f(w) > 0$ for all $i \in [4]$, implying that f is minimized within $w_i \leq z_i$ which is a closed set. Hence there exists $w^* \in \mathbb{R}^4_+$ such that $w_i^* \leq z_i$ and $f(w^*) = \min_{w \in \mathbb{R}^4_+} f(w)$. Now assume $w_i^* > 0$ for all $i \in [4]$, then w^* necessarily is a local minimum and $\nabla^2 f(w^*) \succeq 0$. Conversely, since all local minima of a convex function attain the same value, the statement follows.

The statements for g follow in analogy. □

B.7 Proof of Theorem 7

Proof. The optimal solution to Equation (9) has three possibilities: dense, 3-sparse, 2-sparse.

If the solution to Equation (9) is 2-sparse, $r_{2:4}(w) = 0$ then $w^{(2)}$ is the solution due to that it minimizes $\frac{1}{2}\|w - z\|^2$ subject to the 2-sparse constraint.

If the solution to Equation (9) is 3-sparse with $w_{1:3} > 0$, then $w_{1:3}$ is also the solution to Equation (11), moreover $\nabla^2 g(w_{1:3}) \succeq 0$. Since the constraint is not active, and $w_{1:3}$ is in the strict interior of the constraint, thus $\nabla g(w_{1:3}) = 0$. Since $w_{1:3}$ is feasible, it is also optimal for Equation (13). Any other solution $\tilde{w}_{1:3}$ to Equation (13) (if exists) must satisfy $g(\tilde{w}_{1:3}) = g(w_{1:3})$ thus is also an optimal solution to Equation (11) and Equation (9).

Similarly, if the solution to Equation (9) is dense with $w > 0$ (for all coordinates), then $\nabla^2 f(w) \succeq 0$ and $\nabla f(w) = 0$ (due to stationarity and complementary slackness). This checks the feasibility of w in Equation (14) which implies that w is an optimal solution to Equation (14). Let $\tilde{w} \in \mathbb{R}^4$ be any other optimal solution to Equation (14), $f(\tilde{w}) = f(w)$, thus $\tilde{w}$ is also an optimal solution to Equation (9).

To conclude, in each of the three cases, Line 2-4 of the algorithm returns the optimal solution. □

B.8 Proof of Fact 8

Proof. a) At $w = [0, 0, 0, 0]$ the Hessian Equation (24) is just the identity and hence positive.

b) First note that as soon as one off-diagonal entry of Equation (24) is larger than one, there is a 2×2 principal minor that becomes indefinite, thus the Hessian is not positive semidefinite anymore. Thus positivity of the Hessian implies that all off-diagonal entries are less or equal to 1. We can then use Gershgorin disc theorem and obtain that all eigenvalues are between $[-2, 4]$, hence $\gamma_{\max} \leq 4$. □

C Other Regularizers for 2:4 Sparsity

Our main theoretical contribution is to show that the non-convex proximal operator can be solved efficiently by decomposing it into smaller subproblems. In principle, there also exist simpler regularizers that induce 2:4 sparsity and whose proximal operator can be solved more efficiently. Here we introduce three further proximal operators of simpler regularizers. To be concise, we assume Lemma 3 has already been applied, i.e., $w_1 \geq w_2 \geq w_3 \geq w_4 \geq 0$ and we define the regularizers

$$R_0(w_1, w_2, w_3, w_4) := \begin{cases} 0 & \text{if } w_4 = w_3 = 0, \\ 1 & \text{if } w_4 = 0, w_3 > 0, \\ 2 & \text{if } w_4 > 0. \end{cases} \tag{25}$$

$$R_1(w_1, w_2, w_3, w_4) := w_4 + w_3, \tag{26}$$

$$R_2(w_1, w_2, w_3, w_4) := \frac{1}{2}(w_4^2 + w_3^2). \tag{27}$$

Each of those regularizers equals 0 if and only if the four weights have a 2:4 sparse pattern. So those regularizers do also enforce 2:4 sparsity. Let's look at their proximal operators – again assuming $z_1 \geq z_2 \geq z_3 \geq z_4 \geq 0$.

$$\arg\min_{w \in \mathbb{R}_+^4} f(w), \quad \text{with} \tag{28}$$

$$f(w) := \frac{1}{2}\|w - z\|^2 + \lambda R_p(w_1, w_2, w_3, w_4). \tag{29}$$

Note that the objective of those proximal operators can all be easily be decomposed for each of the variables and trivially $w_1^* = z_1$ and $w_2^* = z_2$. The other optimizations are also simple. For R_0 we have $w_3^* = 0$ if $\lambda > \frac{1}{2}z_3^2$ else z_3; for R_1 we have $w_3^* = \max(z_3 - \lambda, 0)$; For R_2 we have $w_3^* = z_3/(1 + \lambda)$. For w_4^* it follows in analogy for all three cases.

Thus, depending on λ, R_0 forces weights to be zero if they are below a threshold (hard thresholding), R_1 does a soft-thresholding, and R_2 leads to a shrinkage. For large but finite λ, R_0 and R_1 lead to exact $2:4$ sparsity, whilst R_2 does not lead to exact 2:4 sparsity for finite λ.

However, those regularizers commit very early to the sparsity pattern, and none of the behavior of Figure 1 are observed for them. Furthermore, on the toy problem in Section 4.1 they cannot find the optimal solution. We hence focused the main part of this work on the more elaborate proximal operator. For completeness we also report perplexity numbers for it in Table 2 as well as the runtime. Since the optimization problems are trivial and have a close form solution, they are significantly faster, but lead to slightly worse perplexity.

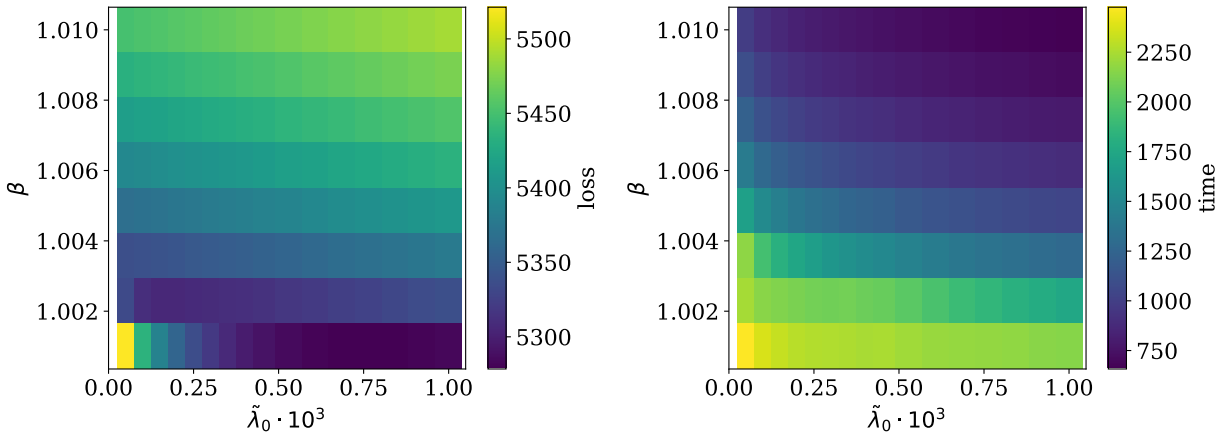

Figure 3: Ablation for the regularizer schedule Equation (15) on the down projection of layer 18 of Llama-3.1 8B.

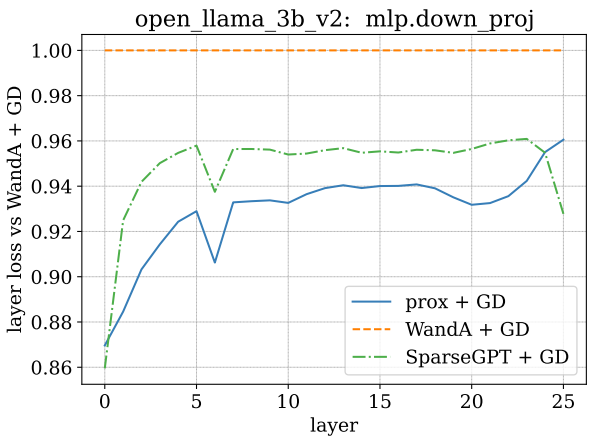

Figure 4: Layerwise local squared loss on an example matrix of openLlama 3bv2. We show the loss relative to the loss WandA (with masked gradient updates) incurs. As intended by design, the proximal approach of pruning leads to smaller local squared loss. In Table 1 we see that this also translates to a better end to end performance in terms of perplexity.

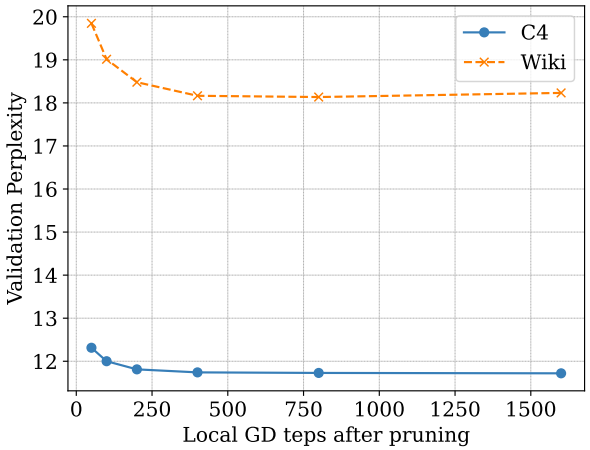

Figure 5: Effect of number of GD steps Equation (5) after masking with Wanda on the 13B model. The perplexity converges quickly and the used 1000 steps in the main paper suffice to drive the local optimization to convergence. These results confirm that the proximal operator identifies a more effective mask, as `prox` achieves lower values of perplexity.

D Experiment Details

In Figures 3, 4, and 5 we show further ablations and insights into our main experiments. Table 2 shows the different runtime of the pruning methods and Table 3 shows results of the pruned models on downstream tasks.

D.1 Using the Hessian of the unpruned Model

For our studies in the main paper we compute the Hessian matrix for each linear layer once before the pruning. This allows us to prune layers in parallel and distribute the pruning on multiple GPUs, because the local Hessians do not depend on each other. The alternative is to prune the matrices sequentially and propagate

Table 2: Wall clock runtime of different pruning approaches on Llama-3.1 8B Instruct. The proposed regularizer leads to the best perplexity, however, also incurs the highest pruning cost. Since this is a once-off effort, and negligibly small compared to pretraining costs, this can be disregarded in production settings.

	Runtime [s]	Runtime [min]	C4 Perplexity	Wiki Perplexity
proposed regularizer [Section 3.2]	11,170	186.17	**26.44**	**20.15**
L0 regularizer, Equation (25)	1,346	22.43	27.25	20.21
L1 regularizer, Equation (26)	1,242	20.70	28.46	20.70
L2 regularizer, Equation (27)	2,628	43.80	27.84	20.43
wanda	**258**	4.30	43.30	26.67
wanda+GD	1,000	16.67	28.45	20.71
sparsegpt	313	5.22	35.55	26.19
sparsegpt+GD	1,064	17.73	27.54	20.85

Table 3: Accuracy Evaluation of pruned versions of Llama-3.1 8B Instruct on 6 Downstream Tasks. The methods (*+GD*) introduced in this work reduce the accuracy gap by around 3% to 5% against previous methods. Note that on GSM8K we see an almost complete breakdown of accuracy. This is because GSM8K is run in a *generative* fashion, where the model can generate many tokens and then provides the final answer in a specific format. It is known that such generative tasks are much more susceptible to accuracy loss than pure *classification* tasks, where the answer options are already provided (Dutta et al., 2024). All tasks other than GSM8K are run as classification tasks, which explains that GSM8K is such an outlier.

model	Mean	MMLU	GSM8K	Winogrande	Hellaswag	TruthfulQA	ai2_arc
original	64.28%	67.70%	75.36%	73.79%	59.19%	37.58%	72.04%
wanda	35.78%	37.02%	2.50%	62.75%	38.94%	26.19%	47.27%
wanda + GD	40.76%	40.65%	11.83%	65.27%	43.98%	27.42%	55.44%
sparsegpt	37.59%	34.97%	6.07%	64.56%	41.50%	24.60%	53.86%
sparsegpt + GD	40.98%	42.94%	8.95%	66.06%	44.74%	27.17%	56.00%
DSnoT	35.01%	36.98%	1.36%	59.75%	36.71%	26.32%	48.96%
prox + GD	40.81%	41.82%	10.01%	65.51%	44.82%	27.29%	55.41%

the pruned inputs. In our ablation Table 4 we find however, that it's effect is overall very minor, and it is not structurally better than using the unpruned inputs.

Table 4: The effect of propagating the pruning errors is negligible.

Model	Method	Hessian	C4	Wiki
3b_v2	dense		9.68	14.15
	wanda	Original	29.49	60.35
	wanda	Pruned	**28.65**	**55.98**
7b_v2	dense		8.84	12.15
	wanda	Original	17.61	27.00
	wanda	Pruned	**17.59**	**26.88**
13b	dense		8.11	11.52
	wanda	Original	**13.17**	**35.39**
	wanda	Pruned	13.30	37.61

D.2 Dependence on the number of calibration samples

In this section, we provide an ablation on the calibration samples for the 3B model, where we consider the same number of samples as in Sun et al. (2024, Table 17)). After pruning we again evaluate the perplexity on the test sets of C4 and Wikitext2. We make three main observations: a/ Wanda (which uses only the

diagonal of the Hessian) converges quickly as known from prior work. b/ Methods that use masked gradient updates can consistently improve their validation perplexity when using more calibration data. However, with diminishing returns (note that the number of samples is increased exponentially here). c/ Except when using only a single sample, our proposed prox+GD outperforms all other methods when using the same amount of calibration data.

Table 5: **C4** perplexity after pruning with varying number of calibration samples

calibration samples	1	16	32	64	128	256	512	1024	2048
wanda	**32.11**	30.20	29.67	29.70	29.66	29.58	29.57	29.52	29.50
wanda + GD	35.72	21.01	19.51	18.99	18.70	18.50	18.36	18.23	18.16
sparsegpt	557.04	21.30	19.92	19.31	19.23	19.01	18.79	18.80	18.64
sparsegpt + GD	302.03	19.77	18.01	17.44	17.12	16.89	16.81	16.74	16.62
prox +GD	70.96	**18.85**	**17.53**	**17.11**	**16.73**	**16.38**	**16.34**	**16.28**	**16.19**

Table 6: **Wiki** perplexity after pruning with varying number of calibration samples

calibration samples	1	16	32	64	128	256	512	1024	2048
Pruning Methods	Wiki	Wiki	Wiki	Wiki	Wiki	Wiki	Wiki	Wiki	Wiki
wanda	**72.89**	63.03	60.16	61.63	61.56	61.51	60.82	60.47	60.54
wanda + GD	100.41	40.92	36.43	34.86	33.96	33.71	33.55	33.05	32.88
sparsegpt	2,859.53	41.01	35.97	33.92	33.60	35.03	33.83	33.95	33.38
sparsegpt + GD	1,675.12	38.06	32.14	30.88	30.13	30.21	30.13	29.56	29.32
prox +GD	149.96	**35.80**	**30.32**	**29.70**	**29.19**	**28.49**	**28.51**	**28.58**	**28.09**

D.3 Prox without masked gradient steps

Since our proposed method intrinsically already uses gradient updates, we highly recommend always adding additional masked gradient steps after freezing the mask. This is particularly advisable given its low computational cost.

However, to understand the effect of it, in Table 7 we also include an ablation on the perplexity as we vary the number of gradient steps after masking. Since prox already does gradient updates during the pruning, even without additional updates after freezing the mask, the perplexity is already fairly low. Doing further masked gradient updates monotonically decreases the perplexity (both in- and out- of distribution) until it reaches convergence.

Table 7: Effect of masked gradient updates for prox on the OpenLlama 3B model.

steps after masking	0	50	100	200	400	800	1000	1600
c4	16.92	16.70	16.56	16.41	16.31	16.27	16.27	16.26
wiki	30.63	29.99	29.58	29.12	28.84	28.66	28.62	28.63