OSSCAR: One-Shot Structured Pruning in Vision and Language Models with Combinatorial Optimization

Xiang Meng [1] **Shibal Ibrahim** [1] **Kayhan Behdin** [1] **Hussein Hazimeh** [2] **Natalia Ponomareva** [2] **Rahul Mazumder** [1]

Abstract

Structured pruning is a promising approach for reducing the inference costs of large vision and language models. By removing carefully chosen structures, e.g., neurons or attention heads, the improvements from this approach can be realized on standard deep learning hardware. In this work, we focus on structured pruning in the one-shot (post-training) setting, which does not require model retraining after pruning. We propose a novel combinatorial optimization framework for this problem, based on a layer-wise reconstruction objective and a careful reformulation that allows for scalable optimization. Moreover, we design a new local combinatorial optimization algorithm, which exploits low-rank updates for efficient local search. Our framework is time and memory-efficient and considerably improves upon state-of-the-art one-shot methods on vision models (e.g., ResNet50, MobileNet) and language models (e.g., OPT-1.3B – OPT-30B). For language models, e.g., OPT-2.7B, *OSSCAR* can lead to $125\times$ lower test perplexity on WikiText with $2\times$ inference time speedup in comparison to the state-of-the-art ZipLM approach. Our framework is also $6\times - 8\times$ faster. Notably, our work considers models with tens of billions of parameters, which is up to $100\times$ larger than what has been previously considered in the structured pruning literature. Our code is available at https://github.com/mazumder-lab/OSSCAR.

1. Introduction

Structured pruning (Lebedev & Lempitsky, 2016; Wen et al., 2016) reduces model size by removing entire subcomponents, e.g., channels in convolutional layers, neurons in dense layers and heads in multi-head attention. Structured pruning offers a practical solution to improve inference latency on standard hardware in contrast to unstructured pruning (LeCun et al., 1989; Hassibi & Stork, 1992; Han et al., 2015), which requires specialized hardware and software.

Despite clear computational benefits from structured pruning, significant challenges remain. Structured pruning is generally associated with a significant drop in utility. To recover lost accuracy, most existing methods (Li et al., 2017; Molchanov et al., 2017; He et al., 2017; Luo et al., 2017; Yu et al., 2018) rely on gradual pruning or iterative retraining, where the model is finetuned on the original loss after every pruning stage. Such finetuning or retraining is expensive—they may not be desirable or possible for large datasets and models under resource constraints. For example, finetuning an LLM on a standard GPU (A100) may not be possible beyond a few billion parameters (Malladi et al., 2023). In this context, recent works such as Kwon et al. (2022); Kurtić et al. (2023) focus on the challenging task of one-shot structured pruning which is performed post-training. Here, a pre-trained model is compressed (without retraining) based on a small amount of calibration data while retaining model accuracy as much as possible.

Despite impressive advances, state-of-the-art methods appear to face challenges in terms of increased computation time, memory usage, and balancing utility with structured sparsity. To address these challenges, we propose a novel optimization-based framework for one-shot structured pruning, which is highly scalable and can achieve good utility-sparsity tradeoffs. We employ a layer-wise reconstruction objective and a careful structure-aware reformulation of the optimization formulation to enable more scalability. To obtain good solutions to the problem, we propose a local combinatorial optimization algorithm, which leverages problem structure to perform efficient local search. By integrating these algorithmic components, we demonstrate that our method is capable of handling large vision and language models, up to 30 billion parameters using a single 32GB V100 GPU — an improvement over prior approaches that can handle up to 340 million parameters.

[1]Massachusetts Institute of Technology, Cambridge, MA, USA [2]Google Research, New York, NY, USA. Correspondence to: Xiang Meng <mengx@mit.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Contributions. Our technical contributions are:

1. Motivated by prior work (He et al., 2017; Dong et al., 2017; Kurtić et al., 2023), we consider a layer-wise reconstruction objective. We formulate the structured pruning problem as a convex quadratic program with combinatorial constraints. Making use of problem-structure (example, by observing that certain groups of variables share the same coefficients), we propose a reformulation of the quadratic objective that significantly reduces the effective size of the Hessian matrix. Our framework, named as *OSSCAR*[1], is a novel general structured pruning framework that prunes channels in convolutional layers, neurons in dense layers or heads in multi-head attention layers.
2. The optimization problem underlying structured pruning is challenging due to its scale, and combinatorial nature. We propose a new efficient algorithm, which performs local combinatorial search to (i) iteratively prune (i.e., zero out) structures to satisfy desired combinatorial constraints, as well as (ii) locally explore structures with similar structured sparsity budgets, but with better objective values. Our algorithm makes heavy use of low-rank matrix updates during its combinatorial search and enjoys theoretical guarantees (runtime and memory cost analysis).
3. *OSSCAR* considerably improves over state-of-the-art approaches for one-shot structured pruning of language and vision models, both in terms of achieved model quality (utility) and inference time. (a) For language models, e.g., OPT-2.7B, *OSSCAR* can lead to $125\times$ lower test perplexity on WikiText with $2\times$ inference time speedup in comparison to state-of-the-art ZipLM (Kurtić et al., 2023) approach. Our framework is also $6\times - 8\times$ faster than ZipLM. *OSSCAR* achieves $1.6\times$ storage reduction in saving decoder layers over ZipLM, when we match performance. (b) For vision models, e.g., ResNet50, *OSSCAR* achieves 10% better accuracy for $\sim 2\times$ speedup over prior state-of-the-art approaches when adapted to the one-shot setting. Our code is publicly available at: `https://github.com/mazumder-lab/OSSCAR`.

It appears that ours is the first work to consider up to 30 billion model sizes for structured pruning, which is $100\times$ larger than what has been considered by prior works on structured pruning (Kwon et al., 2022; Kurtić et al., 2023). The existing state-of-the-art pipelines are unable to scale to structured pruning of model with such sizes.

2. Related Work

Network pruning can be generally categorized into unstructured and structured methods. (a) Unstructured methods (LeCun et al., 1989; Hassibi & Stork, 1992; Han et al., 2015; 2016; Guo et al., 2016) prune unimportant weights in the model, but efficiency of the pruned sparse network cannot be realized on general-purpose GPU hardware. To achieve speedups custom hardware libraries are needed. (b) Structured methods prune channels, neurons etc. (Lebedev & Lempitsky, 2016; Wen et al., 2016), and thus actual speedup can be realized without the need for custom sparse accelerators. In this work, we consider structured pruning. Next, we briefly review structured pruning methods in vision and language models.

Structured pruning in vision models. Lebedev & Lempitsky (2016); Wen et al. (2016); Zhou et al. (2016) consider ℓ_{21} norm to prune filters in convolutional layers. Li et al. (2017) and He et al. (2018a) use ℓ_1 norm and ℓ_2 norm of the filters for pruning respectively. He et al. (2017); Luo et al. (2017) minimize a layer-wise reconstruction error using LASSO and greedy methods to prune channels, respectively. Yu et al. (2018)'s approach minimizes reconstruction error of the final response layer and propagates importance scores through the entire network. Molchanov et al. (2017) uses first-order information from Taylor expansion to greedily prune the least important channels. Liu et al. (2017); Luo & Wu (2020) use scaling factors in Batch Normalization layers as importance scores for pruning channels. Liu et al. (2021) use Fisher-approximation based importance scores to prune channels (or groups of channels). Tang et al. (2020); Lin et al. (2020); Sui et al. (2021) consider feature-importance based measure to determine the important channels. Chen et al. (2021a; 2023) recognize structures that can be pruned without impacting the network output if their parameters are set to zero, and utilize tailored stochastic projected gradient algorithms to efficiently prune such structures. All the above methods are used in the context of gradual pruning, or finetuning, which is prohibitively expensive on large pretrained models and datasets. We explore one-shot pruning adaptations of some of these approaches, and compare them with our approach *OSSCAR*.

Structured pruning in language models. Previous works focus on pruning different components in Transformer models. Voita et al. (2019); Michel et al. (2019) prune heads in multi-head attention layers. McCarley (2019); Hou et al. (2020); Chen et al. (2021b) prune the feed-forward network (FFN) by reducing the intermediate dimension. Fan et al. (2020); Sajjad et al. (2020) drop entire Transformer blocks (a pair of MHA and FFN) from a pre-trained model. Xia et al. (2022); Tao et al. (2023) allow for pruning the hidden dimension shared by all transformer blocks. All of these methods are paired with gradual pruning, or finetuning, which can be prohibitively expensive for large models. Kwon et al. (2022) and Kurtić et al. (2023) consider the more challenging task of post-training one-shot structured pruning, which is the main focus of our work. These approaches

[1] *OSSCAR*: **O**ne-**S**hot **S**tructured **C**ompression **A**lgo**R**ithm

consider pruning both heads in multi-head attention layers and intermediate dimension in FFN blocks. Kwon et al. (2022) considers a local model based on the second-order (Hessian) information of the loss function to prune attention layer heads and hidden neurons in FFN. Although effective, this approach can be prohibitively expensive in terms of runtime and/or memory for billion-parameter models as the ones we consider. Kurtić et al. (2023) consider a layer-wise reconstruction error (He et al., 2017; Luo et al., 2017) to prune attention heads and hidden neurons in FFN. Kwon et al. (2022) and Kurtić et al. (2023) consider BERT-Large and GPT2 models with sizes < 340 million parameters.

Combinatorial optimization in network pruning. Network pruning broadly speaking involves identifying which parameters to set to zero to minimize the impact on performance of the original model. Due to its discrete nature, network pruning can be formulated as a combinatorial optimization problem. Several works have explored unstructured network pruning using combinatorial optimization techniques. Yu et al. (2022) formulates the problem as a mixed-integer quadratic programming problem and employ different heuristics for pruning. Benbaki et al. (2023) considers an ℓ_0 regression formulation and propose algorithms that exploit problem structure for computational benefits. Xu et al. (2023) utilizes combinatorial optimization to determine the optimal sparsity allocation for each layer. Meng et al. (2024) jointly minimizes the number of non-zero parameters and FLOPs using combinatorial optimization. To the best of our knowledge, OSSCAR is a novel framework that directly optimizes the layer-wise structured pruning problem with discrete optimization methods at scale.

Sparsity in linear models. There is a rich body of work on sparsity in linear models: see Hastie et al. (2015); Bach et al. (2012) for an overview of Lasso based procedures. A relatively recent line of work has advanced computational aspects of discrete optimization based procedures for sparse linear models—see Bertsimas et al. (2016); Hazimeh & Mazumder (2020); Hazimeh et al. (2022) for progress in vanilla ℓ_0-sparsity and Hazimeh et al. (2023) (and references therein) for group-sparse settings. The problem of structured pruning in neural networks that we consider here poses interesting and unique challenges compared to the setting of linear models.

Paper organization. The rest of the paper is organized as follows. We present our problem formulation in Section 3 and our proposed algorithm in Section 4. We provide empirical validation of our proposals on large vision and language models in Section 5.

3. Problem Formulation

Here we describe *OSSCAR*: a novel framework for structured pruning under latency constraints. Building on prior work in post-training structured pruning (He et al., 2017; Luo et al., 2017), we adopt a layer-wise pruning strategy. At a high-level our goal here is to prune weights that minimally impact model performance in each layer: i.e., pruned weights w should be comparable to the original pre-trained weights $\widehat{w}$.

Formally, we consider minimizing the squared error loss between the output of a layer with weight $\widehat{w}$ and the pruned weight w (to be learned) over N training samples $\{X^i\}_{i=1}^N$. If $h(w, X)$ denotes the layer's output (before the activation function) with weight w for input X, we minimize the loss

$$L(w) = \frac{1}{2N} \sum_{i=1}^{N} \left\| h(\widehat{w}, X^i) - h(w, X^i) \right\|^2, \qquad (1)$$

subject to some structured sparsity constraints on w.

In all models we consider, $h(w, X)$ is a linear function of w, and $L(w)$ is a quadratic function of w of the form:

$$L(w) = \frac{1}{2} w^\top H' w - G'^\top w. \qquad (2)$$

Here, H' and G' are the quadratic and linear coefficients, respectively. The length of w equals to the number of weights in the layer, potentially reaching hundreds of millions in LLMs (e.g., 200 million weights in the OPT-30B model). This scale presents significant memory and computational challenges in forming/saving H' and evaluating $L(w)$.

To address the computational challenges, we make a key observation: the loss function $L(w)$ is highly structured in the sense that H' is highly sparse, and variables in w can be divided into groups sharing the same quadratic coefficients. This insight allows us to represent w as a matrix (denoted as W) and reformulate the loss as

$$L(W) = \frac{1}{2} \mathrm{Tr}(W^\top H W) - \mathrm{Tr}(G^\top W). \qquad (3)$$

This reformulation significantly downscales the problem: e.g, the quadratic coefficient matrix size can reduce from hundreds of millions to tens of thousands. For instance, with OPT-30B, we only need to handle a $27k \times 27k$ matrix H.

In what follows we discuss how to construct (3) given a network. We consider the case of dense layers, CNNs and LLMs. We show that across all models, our optimization problem can be formulated as a Mixed Integer Quadratic Program (MIQP).

Structured pruning in dense layers. As a motivating example, we start with structured pruning in a dense (linear) layer with an input dimension N_{in} and an output

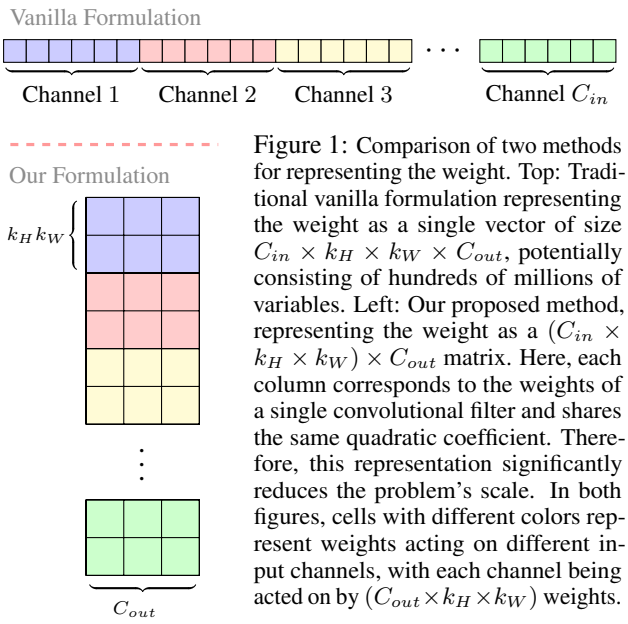


Figure 1: Comparison of two methods for representing the weight. Top: Traditional vanilla formulation representing the weight as a single vector of size $C_{in} \times k_H \times k_W \times C_{out}$, potentially consisting of hundreds of millions of variables. Left: Our proposed method, representing the weight as a $(C_{in} \times k_H \times k_W) \times C_{out}$ matrix. Here, each column corresponds to the weights of a single convolutional filter and shares the same quadratic coefficient. Therefore, this representation significantly reduces the problem's scale. In both figures, cells with different colors represent weights acting on different input channels, with each channel being acted on by $(C_{out} \times k_H \times k_W)$ weights.

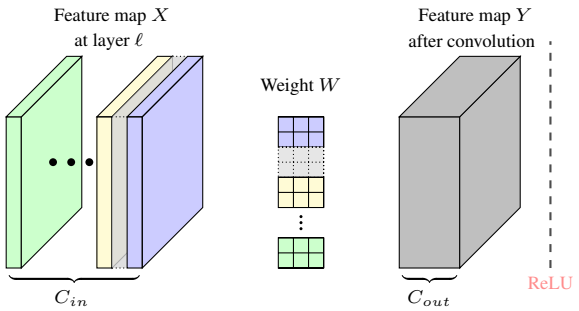


Figure 2: Illustration of structured pruning in a convolutional layer to reduce feature map X's width by pruning weights that act on certain input channels. This figure shows an example where weights acting on the second input channel (denoted by gray cells) are pruned. Consequently, the corresponding channel (also in gray) in the feature map X becomes redundant and can be removed. Practically, we apply an optimization algorithm to prune and update the weight matrix W to minimize the difference between the feature map Y generated from pruned weights and dense weights.

dimension N_{out}. We write W as a $N_{in} \times N_{out}$ matrix, and the input X over N samples as a $N \times N_{in}$ matrix. The loss function can then be expressed as $L(W) = \frac{1}{2}\|X\widehat{W} - XW\|_F^2 = \frac{1}{2}\operatorname{Tr}(W^\top(X^\top X)W) - \operatorname{Tr}((X^\top X\widehat{W})^\top W) + \frac{1}{2}\operatorname{Tr}(\widehat{W}^\top(X^\top X)\widehat{W})$. Consequently, we set $H = X^\top X$ and $G = X^\top X\widehat{W}$ in (3). For structured pruning within dense layers, we would like to remove input neurons, which corresponds to pruning certain rows of the matrix W.

Structured pruning in CNNs. To prune the weights in a convolutional layer, we apply convolutional filters w of size $C_{out} \times C_{in} \times k_H \times k_W$ to an input feature map X of dimensions $C_{in} \times f_h \times f_w$. This feature map X is derived from a data point in the training set. The convolution of w with X produces an output matrix $h(w, X) = \operatorname{Conv}(w, X)$ of size $C_{out} \times f_h \times f_w$. In this notation, C_{in} and C_{out} are the number of input and output channels (the output channel also referred to as a filter), and k_h, k_w, f_h, and f_w represent kernel and feature map dimensions, respectively.

An important insight here is that the outputs of any two filters are independent of each other's weight choices, and their weights share identical quadratic coefficients in (1). Hence we can write the weights as a matrix W with dimensions $(C_{in} \times k_H \times k_W) \times C_{out}$, where each column represents the weights of a single convolutional filter, as illustrated in Fig 1.

This formulation allows us to express the loss $L(W)$ in the form (3). Here, H is a positive semi-definite matrix with dimensions $(C_{in} \times k_H \times k_W) \times (C_{in} \times k_H \times k_W)$, and G is the quadratic problem's linear term, with dimensions $(C_{in} \times k_H \times k_W) \times C_{out}$. Both H and G can be computed[2] using the training samples $\{X^i\}_{i=1}^N$ and the original dense weight $\widehat{W}$.

We adopt a channel pruning approach aimed at reducing the width of feature maps by selectively pruning weights that act on certain input channels (see Fig 2). This method offers two benefits: (i) It enables the removal of redundant input channels affected by the pruned weights, along with the corresponding filters (from the previous layer) that generate these channels. This decreasing inference time. (ii) Rather than removing entire convolutional filters, our approach prunes only parts of each filter. This selective pruning allows for the application of optimization techniques to the unpruned weights, aiming to closely replicate the output of the original filters by minimizing $L(W)$.

As illustrated in Fig. 1, each weight acting on a particular input channel corresponds to a set of rows in the weight matrix W. Thus, to implement channel pruning, it suffices to prune (or remove) some rows of the matrix.

Structured pruning in LLMs. We now discuss structured pruning in Transformers (Zhang et al., 2022). Figure 3 presents the structure of a layer in a Transformer. In line with previous studies (Kwon et al., 2022; Kurtić et al., 2023), our focus is on two structural changes: removing attention heads and reducing the intermediate dimension in fully connected layers. This involves pruning the last linear sublayer in multi-head attention and the second sublayer of the feed-forward network, as highlighted in red in Fig 3.

[2]In practice, we can compute H and G by using e.g, PyTorch's "nn.unfold" function.

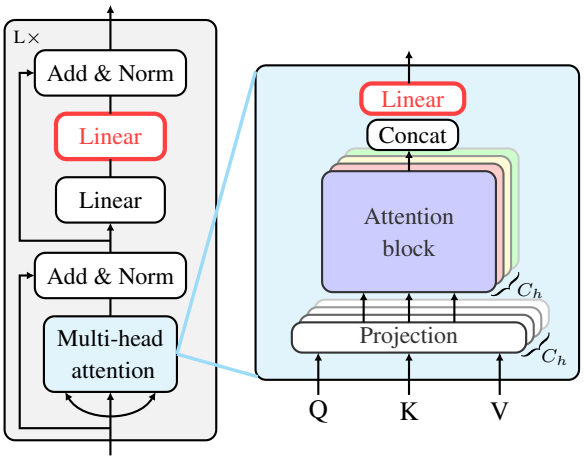


Figure 3: Illustration of structured pruning to accelerate a Transformer layer. The multi-head attention comprises C_h distinct single-head attention blocks. The outputs from these blocks are concatenated and processed through a linear sublayer. We prune the linear sublayer for multi-head integration and the second sublayer of the feed-forward network, as marked in red.

Given that these two sublayers are linear, we can represent W as a $N_{in} \times N_{out}$ matrix, where N_{in} and N_{out} are the input and output dimensions of the linear layer, respectively. Following the discussion on structured pruning in dense layers, we can write loss $L(W)$ in the form (3).

We now describe the structured pruning constraints corresponding to these two types of structural removal:

1. Removing attention heads: the output of each attention head is a D_{head}-dimensional vector, feeding into the multi-head integration linear sublayer. To prune an attention head, we remove D_{head} consecutive rows from the linear sublayer's weight matrix W.
2. Reducing the intermediate dimension in fully connected layers: this involves removing neurons from hidden states, equivalent to removing individual rows from the second linear sublayer's weight matrix. This case aligns with structured pruning for dense layers.

Structured pruning as a MIQP. We now introduce a unified MIQP formulation encompassing structured pruning for all the models discussed previously.

We first divide the rows of the weight matrix W into a partition $Q^1, Q^2, \ldots, Q^p$. To determine p and the partition, we consider three cases:

1. Dense layers: $p = N_{in}$ (input dimension) and each $Q^j = \{j\}$ denotes weights for the j-th input neuron.
2. Convolutional layers: $p = C_{in}$ (the number of input channels) and each partition Q^j includes rows $\{(j-1)k_H k_W + 1, \ldots, jk_H k_W\}$, represents weights acting on the j-th input channel.
3. Attention heads: $p = C_h$ (the number of attention heads) and each partition $Q^j = \{(j-1)D_{head} + 1, \ldots, jD_{head}\}$, represents weights for the j-th head.

We use a binary variable z_j to indicate whether weights in Q_j are kept or pruned. *OSSCAR* aims to prune p' groups of weights while minimizing the loss $L(W)$, and can be described by the following problem (here $\mathbf{w}_i$ denotes the i-th row of matrix W) with both discrete (z) and continuous (W) variables:

$$\begin{aligned} \min_{W,z} \quad & L(W) := \frac{1}{2}\mathrm{Tr}(W^\top H W) - \mathrm{Tr}(G^\top W) \\ \text{s.t.} \quad & \mathbf{w}_i \cdot (1 - z_j) = \mathbf{0}, \ \forall\, j \in [p],\ i \in Q^j \\ & \sum_{j=1}^{p} z_j = p - p', \ \ z_j \in \{0, 1\}. \end{aligned} \tag{4}$$

4. Algorithm

The main goal of this paper is one-shot structured pruning on large-scale models. It is important to obtain high-quality solutions to the MIQP derived from structured pruning both effectively, due to the large scale of the problem, and accurately since we only prune once and will not fine-tune the weights via SGD. To this end, we first reformulate the MIQP (4) (involving both discrete and continuous variables) to a combinatorial problem in Section 4.1 (with only discrete variables). We then introduce an efficient approximate solver based on local search for achieving a high-quality solution in Section 4.2.

4.1. A combinatorial optimization reformulation

The MIQP(4) involves two sets of variables: the binary variables z and the weight matrix W. In practice, z has up to a few thousand variables, while the weight matrix W may have millions of variables. Fixing the binary variables z reduces the problem to a quadratic one, which, despite its large scale, is relatively simple to solve optimally. However, the critical task is to wisely determine the values of z, as they indicate which portions of a layer's input are retained and how effectively the output of the dense weight can be approximated by the pruned weight.

In light of this, we reformulate the MIQP as a combinatorial problem, primarily focusing on optimizing z. We let S denote the set of indices where z is zero as, and define $f(S)$ as the minimum loss $L(W)$ achieved by pruning all rows in Q^j for j in S from the weight matrix W and then updating the remaining weights:

$$\begin{aligned} f(S) = \min_{W} \ & L(W) := \frac{1}{2}\mathrm{Tr}(W^\top H W) - \mathrm{Tr}(G^\top W) \\ \text{s.t.} \ & \mathbf{w}_i = 0, \ \forall\, i \in Q^j,\ j \in S. \end{aligned} \tag{5}$$

The objective is to identify a set S with p' elements that minimizes $f(S)$, and can be written as the following combinatorial problem:

$$\min_{S\subset\{1,2,\ldots,p\}} \quad f(S), \qquad \text{s.t.} \quad |S| = p'. \tag{6}$$

Optimizing $f(S)$ is challenging: potential candidates S selections grow exponentially with the number of channels. For example, pruning half of 32 channels in a CNN layer results in over 600 million possible choices for S. Moreover, calculating the value of $f(S)$ for a specific S involves solving a quadratic problem with up to millions of variables, further complicating this problem. In the next section, we introduce a highly efficient method for identifying a high-quality S.

4.2. Local search based approximate solver

To motivate our local search based approximate solver, we first discuss calculating $f(S)$ for a specific set S. Define

$$I_S := \left\{i \mid i \in Q^j \text{ for some } j \notin S\right\} \tag{7}$$

as the set of rows not constrained to zero in (5). Consequently, computing the value of $f(S)$ becomes a quadratic problem focusing on the weights in rows from I_S:

$$f(S) = \min_{W_{I_S}} \ \frac{1}{2}\mathrm{Tr}(W_{I_S}^\top H_{I_S,I_S} W_{I_S}) - \mathrm{Tr}(G_{I_S}^\top W_{I_S}) \tag{8}$$

Here, H_{I_S,I_S} is the submatrix of H with rows and columns in I_S, and W_{I_S} (G_{I_S}) is the submatrix of W (G) with rows in I_S. The optimal value of this quadratic problem is

$$f(S) = -\frac{1}{2}\,\mathrm{Tr}\left(G_{I_S}^\top (H_{I_S,I_S})^{-1} G_{I_S}\right). \tag{9}$$

The time complexity of computing (9) is $O(d_1^2(d_1+d_2))$, assuming W is a $d_1 \times d_2$ matrix. Solving the combinatorial optimization problem (6) requires evaluating $f(S)$ for numerous distinct sets S. In LLMs, the scales of d_1 and d_2 can reach up to tens of thousands, presenting substantial computational challenges.

To accelerate the evaluation of $f(S)$, a crucial insight is that if we already have $f(S')$ computed for a set S' similar to S, we can apply low-rank matrix updates to efficiently calculate $f(S)$. This leads to the following result:

Proposition 4.1. *Given two sets S and S'. Suppose we have computed the value of $f(S')$, the inverse of $H_{I_{S'},I_{S'}}$ and the optimal weight matrix for $f(S')$. The value of $f(S)$, the inverse of H_{I_S,I_S} and the optimal weight matrix for $f(S)$ can then be computed within $O(td_1(d_1+d_2))$ time complexity, where $t = |I_S \Delta I_{S'}|$ denotes the symmetric difference between I_S and $I_{S'}$.*

Algorithm 1 Local search-based approach for solving (6).

Require: The number of iterations T, two lists $\{t_i\}_{i=1}^T$ and $\{p_i\}_{i=1}^T$ with $\sum_{i=1}^T p_i = p'$ and $p_i \le t_i, \forall i \in [T]$.

1: Initialize with $S_0 = \emptyset$, compute H^{-1} and the optimal weight $W^* = H^{-1}G$.
2: **for** $i = 1, 2, \ldots, T$ **do**
3: Conduct a local search on S_{i-1}: solve problem (10) with $S' = S_{i-1}$, $\hat{t} = t_i$ and $\hat{p} = p_i$ and get S_i.
4: Compute $f(S_i)$, the inverse of $H_{I_{S_i},I_{S_i}}$ and the optimal weight matrix for $f(S_i)$.
5: **end for**
6: **Output:** The set S_T that approximately optimizes (6).

The proof of Proposition 4.1 is in Section A.1. It inspires us to perform a local search at each iteration around the current solution S' by approximately solving:

$$\min_S \ f(S) \quad \text{s.t. } |S\Delta S'| \le \hat{t},\ |S| \ge |S'| + \hat{p}. \tag{10}$$

We restrict the symmetric difference between S' and S by $\hat{t}$. This constraint allows efficient computation of compute $f(S)$, the inverse of H_{I_S,I_S} and the optimal weight matrix for $f(S)$, as per Proposition 4.1. We require $|S| \ge |S'| + \hat{p}$ since our strategy is to start with the empty set and gradually increase the cardinality of S, which means we begin with the dense weight and gradually prune the structural components in the neural network.

To solve (10), we replace elements in S' with minimal impact on the objective with those that have more significant effects. The importance of each element in $[p]$ is evaluated based on the objective's change when the element is added to or removed from S'. This is formally expressed as:

$$B_j = \begin{cases} f(S') - f(S' \setminus \{j\}), & \text{if } j \in S' \\ f(S' \cup \{j\}) - f(S'), & \text{otherwise,} \end{cases} \quad \forall j \in [p]. \tag{11}$$

We set $s_1 = \lfloor \frac{\hat{t}-\hat{p}}{2} \rfloor$ and $s_2 = \lfloor \frac{\hat{t}+\hat{p}}{2} \rfloor$. S_{out} includes the s_1 elements in S' with the lowest B_j values, while S_{in} includes the s_2 elements in $[p] - S'$ with the highest B_j values. Then, we derive the solution to (10) as:

$$S = (S' \setminus S_{out}) \cup S_{in}. \tag{12}$$

The method we propose is concisely outlined in Algorithm 1. Proposition 4.2 details the time and memory complexities of Algorithm 1, with its proof provided in Section A.2.

Proposition 4.2. *Assume each group in the row partition $Q^1, Q^2 \ldots, Q^k$ is of equal size. Then, Algorithm 1 can be executed in $O(Td_1^2(d_1+d_2))$ time and requires $O(d_1(d_1+d_2))$ memory. Furthermore, if we choose $t_i = p_i$ for all $i \in [T]$, then the time complexity reduces to $O(d_1^2(d_1+d_2))$.*

We now discuss the effects of parameters $\hat{t}$ and $\hat{p}$ in (10) on the performance of Algorithm 1. The parameter $\hat{t}$ determines the range of the local search. A larger $\hat{t}$ enables more aggressive updates yet complicates solving (10), reducing the accuracy of our approximate solution to it. Conversely, a smaller $\hat{t}$ leads to precise updates, but this conservative approach requires much more iterations for convergence. As for $\hat{p}$, its larger values (e.g., $\hat{t} = \hat{p}$) make Algorithm 1 resemble a greedy pruning method. In this mode, each iteration adds elements to S with the greatest impact on the objective $f(S)$. On the other hand, a smaller $\hat{p}$ shifts the algorithm towards a local swapping strategy, where elements within S are exchanged with more impactful ones from outside S to find a set with a smaller objective value. Figure 4 provides an illustration of Algorithm 1 under different $\hat{p}$ settings.

In our experiments, we found that the performance of our proposed algorithm is not sensitive to the choice of $\hat{t}$ and $\hat{p}$. Therefore, we do not tune over these parameters much, and we set $\hat{t} = \hat{p} \leq 10$, with the precise values of $\hat{t}$ and $\hat{p}$ depending on the problem size (detailed in Section B.1). We also conduct an ablation study to examine the algorithm's performance with various choices of $\hat{t}$ and $\hat{p}$ and show the low sensitivity of our algorithm to these hyperparameters. The results are presented in Appendix B.2.5.

5. Numerical Experiments

In this section, we compare our proposed framework *OSSCAR* with leading structured pruning methods in vision and language models. We provide detailed information on the experimental setup and reproducibility in Appendix B.1. Additional experimental results and ablation studies are given in Appendix B.2.

Our proposed framework *OSSCAR* employs a one-shot pruning method, which prunes weights just once without finetuning. On the other hand, some structured pruning methods (Li et al., 2017; Sui et al., 2021; Luo et al., 2017) perform a fine-tuning procedure after pruning to retrain the weights via stochastic gradient descent. One-shot pruning is more efficient than fine-tuning, which typically requires a significantly larger number of training samples and greater computational resources. For instance, fine-tuning ResNet50 on ImageNet involves over a million training samples and days of training, whereas our one-shot pruning experiments only take a few minutes and 500 training samples. Additionally, fine-tuning large language models often becomes impractical due to the high demand for computational resources. Therefore, in our experiments, we focus on one-shot pruning. To ensure fairness in our comparisons, we also consider the performance of competing methods in one-shot setting.

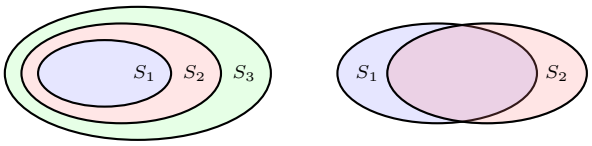


Figure 4: The structure of sets $\{S_i\}_{i=1}^T$ under different choices of $\hat{p}$. Left: with a large $\hat{p}$, Algorithm 1 mimics a greedy pruning approach, incrementally expanding the set S and resulting in nested sets $S_1 \subset S_2 \subset S_3 \cdots$. Right: with a small $\hat{p}$, Algorithm 1 employs a local swapping strategy, leading to sets without a nested structure.

5.1. Structured pruning in vision models

We consider various pre-trained convolutional networks including ResNet20 (He et al., 2016, 260k parameters) trained on CIFAR10 (Krizhevsky et al., 2009), MobileNet ((Howard et al., 2017), 4.2M parameters) and ResNet50 (25.6M parameters) trained on ImageNet (Deng et al., 2009). We assess the performance of all structured pruning methods using 500 training samples as the calibration dataset.

Competing methods. We compare it with several one-shot pruning methods in vision models: (i) Magnitude Pruning (MP) (Mozer & Smolensky, 1989; He et al., 2018a), (ii) CHIP (Sui et al., 2021), (iii) FPGM (He et al., 2018b), (iv) Lasso (He et al., 2017) and (v) ThiNet (Luo et al., 2017). The configuration details for these methods and the parameter settings for *OSSCAR* are outlined in Section B.1.1.

Performance on a single layer. We first evaluate structured pruning on a single layer. Figure 5 compares the squared loss between dense and pruned weights at different speedup ratios. *OSSCAR* significantly outperforms MP, CHIP, and FPGM, as these methods do not directly minimize $L(W)$. Compared to ThiNet and Lasso, which aim to minimize $L(W)$, *OSSCAR* achieves about 20% lower loss under the same speedup ratios. These results demonstrate the effectiveness of our proposed local optimization method (Algorithm 1) in obtaining high-quality solutions for the layerwise channel pruning problem.

Accuracy performance. The effectiveness of Algorithm 1 in solving pruning problems at each layer translates to improved out-of-sample performance of the overall pruned model, as demonstrated in Table 1. The table compares the test accuracy for pruned ResNet20, MobileNetV1, and ResNet50 across different speedup ratios. *OSSCAR* consistently achieves higher accuracy compared to existing methods, particularly at higher speedup ratios, with $15\% \sim 30\%$ higher accuracy. These results showcase the superiority of *OSSCAR* in maintaining high accuracy while achieving significant speedups.

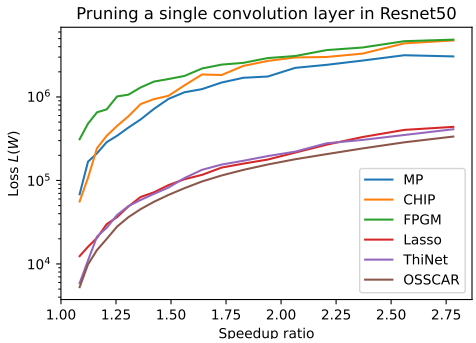


Figure 5: Performance of pruning the layer "layer1.0.conv2" in Resnet50 at different speedup ratios. The plot shows the squared loss between dense and pruned weights of this single layer, using different pruning methods. The speedup ratio is computed as the total number of channels (64) divided by the number of channels retained after pruning.

Table 1: One-shot structured pruning performance (accuracy) of various methods on ResNet20, MobileNetV1, and ResNet50. The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods, we take ten runs and report the mean.

Model	Speedup	MP	CHIP	FPGM	Lasso	ThiNet	*OSSCAR*
ResNet20 on CIFAR10 (91.36%)	1.3x	55.37	67.24	39.01	89.03	89.16	**89.20** (±0.12)
	1.4x	39.54	39.05	24.37	86.68	87.09	**87.70** (±0.16)
	1.7x	26.92	19.16	12.82	83.55	84.11	**85.05** (±0.19)
	2.0x	18.98	11.81	10.92	78.80	79.30	**81.10** (±0.49)
	2.6x	11.04	9.97	12.45	68.58	70.35	**72.60** (±0.78)
	3.5x	10.21	10.31	12.60	57.15	58.68	**61.82** (±0.98)
MobileNetV1 on ImageNet (71.95%)	1.3x	21.66	33.16	4.42	70.13	70.07	**70.70** (±0.05)
	1.4x	1.60	5.36	0.58	67.65	67.66	**68.89** (±0.08)
	1.5x	0.13	0.77	0.11	63.79	64.06	**66.37** (±0.15)
	1.7x	0.10	0.22	0.10	58.88	59.51	**63.03** (±0.31)
	1.9x	0.10	0.10	0.13	52.72	53.34	**58.39** (±0.36)
	2.2x	0.10	0.10	0.10	45.47	45.77	**52.07** (±0.57)
ResNet50 on ImageNet (77.01%)	1.2x	60.57	57.06	59.53	73.82	73.49	**74.33** (±0.08)
	1.3x	9.28	16.86	18.79	67.07	66.29	**69.30** (±0.16)
	1.4x	3.28	5.34	5.48	61.58	60.44	**65.32** (±0.27)
	1.6x	0.95	1.24	1.42	53.15	51.91	**59.14** (±0.44)
	1.7x	0.37	0.57	0.40	41.68	41.08	**50.16** (±0.59)
	1.9x	0.25	0.29	0.21	29.23	28.87	**38.45** (±0.63)

5.2. Structured Pruning in Large Language Models

Next, we evaluate the usefulness of our proposed framework *OSSCAR* for one-shot structured pruning on LLMs.

Models and datasets. We focus on pruning the OPT model family (Zhang et al., 2022), with sizes ranging from 1.3 billion to 30 billion parameters. For calibration data, we adopt the approach of Frantar & Alistarh, 2023, utilizing 128 segments of 2048 tokens each, randomly selected from the first shard of the C4 dataset (Raffel et al., 2020). We consider perplexity as our key metric, recognized for its challenge and stability in evaluating the performance of pruned models (Yao et al., 2022; Xiao et al., 2023; Frantar & Alistarh, 2023). The perplexity is calculated following precisely the procedure described by HuggingFace (Per, 2022), using full stride. We consider the test sets of raw-WikiText2 (Merity et al., 2017) and PTB (Marcus et al., 1994) as well as a subset of the C4 validation data, all popular benchmarks in LLM pruning literature (Yao et al., 2022; Xiao et al., 2023; Frantar & Alistarh, 2023). We use the HuggingFace Transformers library (Wolf et al., 2020) for handling the models and datasets. In addition to perplexity, we evaluate the pruned models on 4 different zero-shot tasks: Piqa (Bisk et al., 2020), Lambada (Paperno et al., 2016), Arc easy, and challenge (Clark et al., 2018), and report results in Section B.2.2.

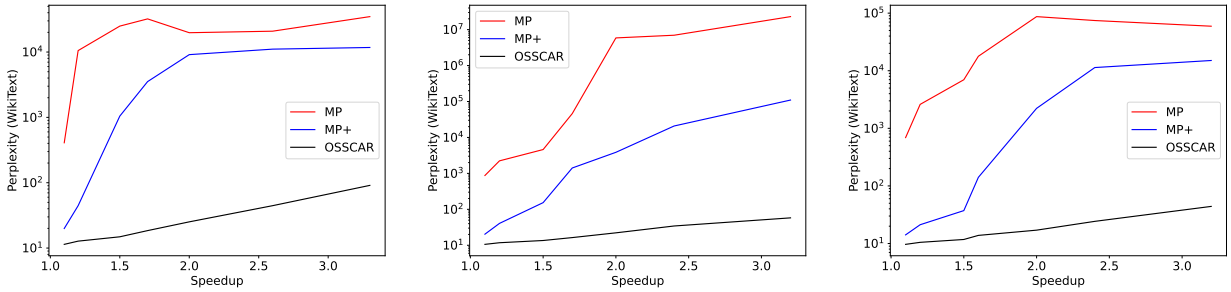
Competing methods. We compare against multiple one-shot structured pruning methods: (i) Magnitude pruning (MP), (ii) Magnitude pruning with a layer-wise refinement step (MP+), and (iii) ZipLM[3] (Kurtić et al., 2023). The configuration details for these methods, along with the parameter settings for *OSSCAR*, are outlined in Section B.1.2.

Results. We report the perplexity performance on the raw-WikiText2 test set across various speedup ratios for different models (OPT-1.3B, OPT-2.7B), as shown in Table 2. Interestingly, we observe a rapid increase in perplexity for pruned models using baseline methods, even at relatively small speedup ratios. *OSSCAR*, in contrast, demonstrates substantial gains over baseline methods. Notably, it shows a much smaller increase in perplexity with higher speedups. For instance, *OSSCAR* achieves a perplexity 125 times lower than the state-of-the-art ZipLM framework for one-shot structured pruning for the OPT 2.7 billion parameter model at a 2x inference speedup.

[3]The authors of ZipLM (state-of-the-art) only considered models of sizes < 340 million. We apply the ZipLM framework to OPT models. For the second stage of their framework (structured SPDY search), we turn off the randomized search component, as it appears to be computationally intractable.

Table 2: Perplexity performance on Wikitext for one-shot structured pruning of OPT models (1.3B, 2.7B, and 6.7B). The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods we take ten runs and report the mean and standard error.

Model	Speedup	MP	MP+	ZipLM	*OSSCAR*
OPT-1.3B	1.2x	163.0 (±0.20)	22.01 (±0.09)	**14.58** (±0.08)	15.54 (±0.15)
	1.3x	1834 (±1.00)	41.46 (±0.28)	61.36 (±5.19)	**17.53** (±0.11)
	1.4x	7412 (±29.0)	4953 (±151)	729.0 (±114)	**20.49** (±0.36)
	1.7x	8752 (±55.0)	4802 (±287)	1829 (±66.0)	**25.74** (±0.54)
	2.0x	8439 (±15.0)	6490 (±113)	3529 (±346)	**37.87** (±0.64)
	2.6x	9546 (±4.00)	7608 (±241)	6424 (±301)	**68.50** (±1.89)
	3.3x	12006 (±50.0)	8772 (±256)	9424 (±761)	**153.0** (±3.56)
OPT-2.7B	1.2x	234.2 (±3.07)	22.33 (±0.12)	**12.14** (±0.03)	13.14 (±0.17)
	1.3x	3817 (±108)	46.31 (±0.59)	21.83 (±3.47)	**14.94** (±0.22)
	1.4x	6957 (±794)	111.2 (±2.50)	414.9 (±113)	**17.11** (±0.21)
	1.7x	13903 (±208)	9977 (±5025)	1820 (±422)	**21.18** (±0.24)
	2.0x	12793 (±48.0)	12003 (±2419)	3611 (±840)	**29.48** (±0.46)
	2.4x	11975 (±18.0)	15979 (±3180)	9209 (±2356)	**51.90** (±1.06)
	3.0x	15874 (±62.0)	12433 (±596)	14039 (±1863)	**102.7** (±3.24)



OPT-6.7B OPT-13B OPT-30B

Figure 6: Perplexity performance on WikiText (in log-scale) for one-shot structured pruning of OPT models (6.7B, 13B, and 30B). The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods, we take ten runs and report the mean perplexity.

Performance on larger OPT Models We next showcase the one-shot pruning performance of *OSSCAR* on larger OPT Models (6.7B, 13B, and 30B). Due to memory limitations, ZipLM (Kurtić et al., 2023) cannot process these models[4]. Thus, we compare our framework solely against magnitude pruning-based methods (MP and MP+). Figure 6 displays the perplexity for these models on raw-WikiText2, with the logarithmic scale of the y-axis highlighting that *OSSCAR* reduces the perplexity by at least two orders of magnitude against the baselines. *OSSCAR*'s superior performance stems from achieving much smaller layer-wise reconstruction error at each layer compared to MP and MP+, resulting in less impact on the model's performance.

The overall loss in perplexity from one-shot structured pruning can be mitigated by increasing the sample size from 128 to 2048. For instance, at a 2x speedup, this adjustment can reduce perplexity by up to 3.5 points. We explored this in an ablation study detailed in the Supplementary Section B.2.3.

Timing comparison. Our framework stands out for its time efficiency in pruning networks. Table 3 displays the total time different algorithms take to prune a model to various speed-up ratios. As shown, *OSSCAR* is notably faster, achieving speeds 6–8 times faster than the ZipLM framework. Remarkably, our optimization framework is only 1.5x slower than MP+ baseline, yet it offers substantial improvements in predictive performance.

Table 3: Total time taken by different algorithms to prune a model to various speed-up ratios. For all methods we take ten runs and report the mean and standard error.

Model	MP	MP+	ZipLM	*OSSCAR*
OPT-1.3B	0.469 (±0.00)	1166 (±3.00)	9161 (±670)	1455 (±4.00)
OPT-2.7B	0.996 (±0.00)	2229 (±8.00)	22839 (±999)	2842 (±12.0)
OPT-6.7B	2.551 (±0.01)	5084 (±67.0)	-	7434 (±96.0)

6. Conclusion

We introduce *OSSCAR*, a novel optimization framework for one-shot structured pruning in large-scale vision and language models. *OSSCAR* is based on a reformulation of the layer-wise reconstruction objective, which exploits problem structure to allow for scalable optimization. We develop a novel local combinatorial optimization algorithm that exploits low-rank updates for efficient local search. Our framework is both time- and memory-efficient, markedly enhancing the practicality and performance of one-shot structured pruning methods. For example, on OPT-2.7B, *OSSCAR* can lead to $125\times$ lower test perpexity on WikiText with $2\times$ inference time speedup in comparison to state-of-the-art ZipLM approach. Our pruning framework takes $6\times$ – $8\times$ lesser time to prune the network. Our framework can also prune $100\times$ larger models than previous state-of-the-art structured pruning frameworks.

[4]The original ZipLM pipeline runs out of memory for 1.3B and 2.7B models as well. We added CPU offloading (see Section B.1.2), allowing it to work with 1.3B and 2.7B models.

Acknowledgements

This research is supported in part by grants from ONR (N000142112841 and N000142212665). We acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the research results reported within this paper. Additionally, we thank Google for providing us with Google Cloud Credits to run some of the computational experiments reported in this paper. We thank Wenyu Chen and Riade Benbaki for helpful discussions.

Impact Statement

Structured Pruning approaches are used to improve the efficiency of large models for inference. We do not see direct, negative societal impacts from this research.

References

Perplexity of fixed-length models, 2022. URL https://huggingface.co/docs/transformers/perplexity.

Bach, F., Jenatton, R., Mairal, J., and Obozinski, G. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, 2012.

Benbaki, R., Chen, W., Meng, X., Hazimeh, H., Ponomareva, N., Zhao, Z., and Mazumder, R. Fast as chita: Neural network pruning with combinatorial optimization. In *International Conference on Machine Learning*, pp. 2031–2049. PMLR, 2023.

Bertsimas, D., King, A., and Mazumder, R. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, 2016.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Chen, T., Ji, B., Ding, T., Fang, B., Wang, G., Zhu, Z., Liang, L., Shi, Y., Yi, S., and Tu, X. Only train once: A one-shot neural network training and pruning framework. *Advances in Neural Information Processing Systems*, 34: 19637–19651, 2021a.

Chen, T., Liang, L., Ding, T., Zhu, Z., and Zharkov, I. Otov2: Automatic, generic, user-friendly. *arXiv preprint arXiv:2303.06862*, 2023.

Chen, X., Cheng, Y., Wang, S., Gan, Z., Wang, Z., and Liu, J. EarlyBERT: Efficient BERT training via early-bird lottery tickets. In Zong, C., Xia, F., Li, W., and Navigli, R. (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 2195–2207, Online, August 2021b. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.171. URL https://aclanthology.org/2021.acl-long.171.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dong, X., Chen, S., and Pan, S. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Frantar, E. and Alistarh, D. SparseGPT: Massive language models can be accurately pruned in one-shot. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/frantar23a.html.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 1387–1395, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pp. 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Hassibi, B. and Stork, D. Second order derivatives for network pruning: Optimal brain surgeon. In Hanson, S., Cowan, J., and Giles, C. (eds.), *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

Hastie, T., Tibshirani, R., and Wainwright, M. *Statistical learning with sparsity: the lasso and generalizations*. CRC press, 2015.

Hazimeh, H. and Mazumder, R. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Operations Research*, 68(5):1517–1537, 2020.

Hazimeh, H., Mazumder, R., and Saab, A. Sparse regression at scale: Branch-and-bound rooted in first-order optimization. *Mathematical Programming*, 196(1-2):347–388, 2022.

Hazimeh, H., Mazumder, R., and Radchenko, P. Grouped variable selection with discrete optimization: Computational and statistical perspectives. *The Annals of Statistics*, 51(1):1–32, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1398–1406, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society. doi: 10.1109/ICCV.2017.155. URL https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.155.

He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 2234–2240. AAAI Press, 2018a. ISBN 9780999241127.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4335–4344, 2018b.

Hou, L., Huang, Z., Shang, L., Jiang, X., Chen, X., and Liu, Q. Dynabert: Dynamic bert with adaptive width and depth. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kurtić, E., Frantar, E., and Alistarh, D. Ziplm: Inference-aware structured pruning of language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=d8j3lsBWpV.

Kwon, W., Kim, S., Mahoney, M. W., Hassoun, J., Keutzer, K., and Gholami, A. A fast post-training pruning framework for transformers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=0GRBKLBjJE.

Lebedev, V. and Lempitsky, V. Fast convnets using group-wise brain damage. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2554–2564, 2016. doi: 10.1109/CVPR.2016.280.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., and Shao, L. Hrank: Filter pruning using high-rank feature map. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1526–1535, Los Alamitos, CA, USA, jun 2020. IEEE Computer Society. doi: 10.1109/CVPR42600.2020.00160. URL https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.00160.

Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J.-H., Wang, X., Chen, Y., Yang, W., Liao, Q., and Zhang, W. Group fisher pruning for practical network compression. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7021–7032. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/liu21ab.html.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2755–2763, 2017. doi: 10.1109/ICCV.2017.298.

Luo, J., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 5068–5076. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.541.

Luo, J.-H. and Wu, J. Neural network pruning with residual-connections and limited-data. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1455–1464, 2020. doi: 10.1109/CVPR42600.2020.00153.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Vota6rFhBQ.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pp. 114–119, USA, 1994. Association for Computational Linguistics. ISBN 1558603573. doi: 10.3115/1075812.1075835. URL https://doi.org/10.3115/1075812.1075835.

McCarley, J. S. Pruning a bert-based question answering model. *ArXiv*, abs/1910.06360, 2019. URL https://api.semanticscholar.org/CorpusID:204575977.

Meng, X., Chen, W., Benbaki, R., and Mazumder, R. Falcon: Flop-aware combinatorial optimization for neural network pruning. In *International Conference on Artificial Intelligence and Statistics*, pp. 4384–4392. PMLR, 2024.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Michel, P., Levy, O., and Neubig, G. Are sixteen heads really better than one? In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJGCiw5gl.

Mozer, M. C. and Smolensky, P. Using relevance to reduce network size automatically. *Connection Science*, 1(1): 3–16, 1989.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435.

Sajjad, H., Dalvi, F., Durrani, N., and Nakov, P. On the effect of dropping layers of pre-trained transformer models. *Comput. Speech Lang.*, 77:101429, 2020. URL https://api.semanticscholar.org/CorpusID:251005814.

Sui, Y., Yin, M., Xie, Y., Phan, H., Zonouz, S. A., and Yuan, B. CHIP: CHannel independence-based pruning for compact neural networks. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=EmeWbcWORRg.

Tang, Y., Wang, Y., Xu, Y., Tao, D., Xu, C., Xu, C., and Xu, C. Scop: Scientific control for reliable neural network pruning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Tao, C., Hou, L., Bai, H., Wei, J., Jiang, X., Liu, Q., Luo, P., and Wong, N. Structured pruning for efficient generative pre-trained language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 10880–10895, 2023.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL https://aclanthology.org/P19-1580.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2082–2090, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Wolf, T., Debut, L., Sanh, V., et al. Transformers: State-of-the-art natural language processing. In Liu, Q. and Schlangen, D. (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

Xia, M., Zhong, Z., and Chen, D. Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*, 2022.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Xu, K., Wang, Z., Geng, X., Wu, M., Li, X., and Lin, W. Efficient joint optimization of layer-adaptive weight pruning in deep neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17447–17457, 2023.

Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., and He, Y. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27168–27183. Curran Associates, Inc., 2022.

Yu, R., Li, A., Chen, C., Lai, J., Morariu, V. I., Han, X., Gao, M., Lin, C., and Davis, L. S. Nisp: Pruning networks using neuron importance score propagation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9194–9203, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society. doi: 10.1109/CVPR.2018.00958. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00958.

Yu, X., Serra, T., Ramalingam, S., and Zhe, S. The combinatorial brain surgeon: pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pp. 25668–25683. PMLR, 2022.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zhang, X., Zou, J., He, K., and Sun, J. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015.

Zhou, H., Alvarez, J. M., and Porikli, F. Less is more: Towards compact cnns. In Leibe, B., Matas, J., Sebe, N., and Welling, M. (eds.), *Computer Vision – ECCV 2016*, pp. 662–677, Cham, 2016. Springer International Publishing.

Supplement

A. Proofs of Main Results

A.1. Proof of Proposition 4.1

We first establish some key notations. For a given matrix H and two subsets of indices I_1 and I_2, H_{I_1,I_2} represents the submatrix of H that includes rows indexed by I_1 and columns indexed by I_2; $H_{I_1,:}$ represents the submatrix of H containing only the rows in I_1; $H_{:,I_2}$ represents the submatrix of H containing only the columns in I_2. Moreover, $\mathbf{E}_a$ represents the identity matrix of size $a \times a$, and $\mathbf{0}_{a\times b}$ refers to the zero matrix of dimensions $a \times b$. For any given set $S \subset [p]$, we denote $t_S = |I_S|$, where $I_S = \left\{i \mid i \in C^j \text{ for some } j \notin S\right\}$.

Let us assume we have two sets S and S' with $t = |I_S \Delta I_{S'}|$. We have already calculated the quadratic coefficients $H \in \mathbb{R}^{d_1\times d_1}$ and linear coefficients $G \in \mathbb{R}^{d_1\times d_2}$ in (5). Additionally, we've computed the value of

$$f(S') = -\frac{1}{2}\operatorname{Tr}\left(G^\top_{I_{S'},:}(H_{I_{S'},I_{S'}})^{-1}G_{I_{S'},:}\right), \tag{13}$$

the inverse of $H_{I_{S'},I_{S'}}$, and the optimal weight matrix for $f(S')$, denoted as $W^{(S')}$. The remaining part of the proof will demonstrate that the value of $f(S)$, the inverse of H_{I_S,I_S} and the optimal weight matrix $W^{(S)}$ for $f(S)$ can be computed with a time complexity of $O(td_1(d_1+d_2))$.

Based on the inclusion relationship of sets I_S and $I_{S'}$, we consider the following *three* cases.

Case 1: $I_S \subset I_{S'}$.

In this case we have $t = |I_r| := |I_{S'} \backslash I_S|$. Without loss of generality, we can rearrange the rows and columns of $H_{I_{S'},I_{S'}}$ such that $H_{I_{S'},I_{S'}}$ and its inverse are structured as follows:

$$H_{I_{S'},I_{S'}} = \left[\begin{array}{c:c} H_{I_S,I_S} & H_{I_S,I_r} \\ \hdashline H_{I_r,I_S} & H_{I_r,I_r} \end{array}\right], \ \left(H_{I_{S'},I_{S'}}\right)^{-1} = \left[\begin{array}{c:c} A & B \\ \hdashline B^\top & C \end{array}\right], \tag{14}$$

where A is a submatrix of $\left(H_{I_{S'},I_{S'}}\right)^{-1}$ with the same size as H_{I_S,I_S}, B has the same size as H_{I_S,I_r}, and C has the same size as H_{I_r,I_r}. Note that we have computed the $\left(H_{I_{S'},I_{S'}}\right)^{-1}$, so we have the values of A, B and C.

From the equation

$$H_{I_{S'},I_{S'}}\left(H_{I_{S'},I_{S'}}\right)^{-1} = \mathbf{E}_{t_{S'}} \tag{15}$$

it follows that

$$H_{I_S,I_S}A + H_{I_S,I_r}B^\top = \mathbf{E}_{t_S} \tag{16}$$

$$H_{I_S,I_S}B + H_{I_S,I_r}C = \mathbf{0}_{t_S\times t}. \tag{17}$$

From (17), we deduce

$$H_{I_S,I_r} = -H_{I_S,I_S}BC^{-1}, \tag{18}$$

and substituting this into (16) yields

$$(H_{I_S,I_S})^{-1} = A - BC^{-1}B^\top. \tag{19}$$

Given that B is a $t_S \times t$ matrix, and C is a $t \times t$ matrix, computing $= A - BC^{-1}B^\top$ requires $O(t^3 + t_S^2 t) = O(td_1(d_1+d_2))$ complexity.

Next, we focus on computing $W^{(S)}$. This is the optimal weight matrix that resolves the following quadratic problem:

$$\begin{aligned} \min_W \ & L(W) = \frac{1}{2}\mathrm{Tr}(W^\top HW) - \mathrm{Tr}(G^\top W) \\ \text{s.t.} \quad & \mathbf{w}_i = 0, \ \forall\, i \in Q^j, \ j \in S. \end{aligned} \tag{20}$$

This quadratic problem can be solved analytically, yielding the optimal solution $W^{(S)}$ as:

$$W^{(S)}_{I_S,:} = (H_{I_S,I_S})^{-1} G_{I_S,:}, \tag{21}$$

where all elements in $W^{(S)}$ that do not belong to rows in I_S are zero. Similarly, we can express $W^{(S')}$ as

$$W^{(S')}_{I_{S'},:} = \left(H_{I_{S'},I_{S'}}\right)^{-1} G_{I_{S'},:} \tag{22}$$

where all elements in $W^{(S')}$ that do not belong to rows in $I_{S'}$ are zero. We may further simplify the expression of $W^{(S')}$ as

$$\begin{aligned} W^{(S')}_{I_{S'},:} &= \begin{bmatrix} W^{(S')}_{I_S,:} \\ \hdashline W^{(S')}_{I_r,:} \end{bmatrix} = \left[\begin{array}{c:c} A & B \\ \hdashline B^\top & C \end{array}\right] \begin{bmatrix} G_{I_S,:} \\ \hdashline G_{I_r,:} \end{bmatrix} = \begin{bmatrix} AG_{I_S,:} + BG_{I_r,:} \\ \hdashline B^\top G_{I_S,:} + CG_{I_r,:} \end{bmatrix} \\ &= \begin{bmatrix} (A - BC^{-1}B^\top)G_{I_S,:} + \left(BG_{I_r,:} + BC^{-1}B^\top G_{I_S,:}\right) \\ \hdashline B^\top G_{I_S,:} + CG_{I_r,:} \end{bmatrix} = \begin{bmatrix} W^{(S)}_{I_S,0} \\ \hdashline \mathbf{0}_{t\times d_2} \end{bmatrix} + \begin{bmatrix} BC^{-1}W^{(S')}_{I_r,:} \\ \hdashline W^{(S')}_{I_r,:} \end{bmatrix} \end{aligned} \tag{23}$$

This implies that we can compute $W^{(S)}$ as

$$W^{(S)}_{I_{S'},:} = \begin{bmatrix} W^{(S)}_{I_S,0} \\ \hdashline \mathbf{0}_{t\times d_2} \end{bmatrix} = W^{(S')}_{I_{S'},:} - \begin{bmatrix} BC^{-1} \\ \hdashline \mathbf{E}_{t\times d_2} \end{bmatrix} W^{(S')}_{I_r,:}. \tag{24}$$

Given that B is a $t_S \times t$ matrix, C is a $t \times t$ matrix, and $W^{(S')}_{I_r,:}$ is a $t \times d_2$ matrix, computing W^S requires $O(t_S t d_2) = O(t d_1(d_1 + d_2))$ time complexity.

Finally, we compute the value of $f(S)$. Given that the optimal weight matrix for $f(S')$ is $W^{(S')}$, we can express

$$f(S') = \frac{1}{2}\operatorname{Tr}\left((W^{(S')})^\top H W^{(S')}\right) - \operatorname{Tr}\left(G^\top W^{(S')}\right). \tag{25}$$

It follows a similar argument that

$$f(S) = \frac{1}{2}\operatorname{Tr}\left((W^{(S)})^\top H W^{(S)}\right) - \operatorname{Tr}\left(G^\top W^{(S)}\right). \tag{26}$$

Together with (24), we obtain by some calculations that

$$\begin{aligned} f(S') &= \frac{1}{2}\operatorname{Tr}\left((W^{(S')})^\top H W^{(S')}\right) - \operatorname{Tr}\left(G^\top W^{(S')}\right) \\ &= \frac{1}{2}\operatorname{Tr}\left((W^{(S')}_{I_{S'},:})^\top H_{I_{S'},I_{S'}} W^{(S')}_{I_{S'},:}\right) - \operatorname{Tr}\left(G^\top_{I_{S'},:} W^{(S')}_{I_{S'},:}\right) \\ &= f(S) + \frac{1}{2}\operatorname{Tr}\left((W^{(S')}_{I_r,:})^\top \left[\begin{array}{c:c} BC^{-1} & \mathbf{E}_{t\times d_2} \end{array}\right] H_{I_{S'},I_{S'}} \begin{bmatrix} BC^{-1} \\ \hdashline \mathbf{E}_{t\times d_2} \end{bmatrix} W^{(S')}_{I_r,:} \right) \\ &\quad + \operatorname{Tr}\left((W^{(S')}_{I_r,:})^\top \left[\begin{array}{c:c} BC^{-1} & \mathbf{E}_{t\times d_2} \end{array}\right] \left(H_{I_{S'},I_{S'}} W^{(S')}_{I_{S'},:} - G_{I_{S'},:} \right) \right) \\ &= f(S) + \frac{1}{2}\operatorname{Tr}\left((W^{(S')}_{I_r,:})^\top C^{-1} W^{(S')}_{I_r,:} \right) \end{aligned} \tag{27}$$

Given that C is a $t \times t$ matrix, and $W^{(S')}_{I_r,:}$ is a $t \times d_2$ matrix, computing $f(S)$ requires $O(t^2 d_2)$ time complexity.

We emphasize that, unlike Cases 2 and 3, computing $f(S)$ in this case only requires a time complexity of $O(t^2 d_2)$, as opposed to $O(t d_1(d_1 + d_2))$. This key observation aids in reducing the time cost of Algorithm 1 under certain choices of parameters. We formally present this observation as a lemma:

Lemma A.1. *Given two sets S and S' such that $I_S \subset I_{S'}$ and $t = |I_{S'} \backslash I_S|$. Suppose we have computed the value of $f(S')$, the inverse of $H_{I_{S'},I_{S'}}$ and the optimal weight matrix for $f(S')$. The value of $f(S)$ can then be computed within $O(t^2 d_1)$ time complexity.*

Case 2: $I_{S'} \subset I_S$.

In this case we have $t = |I_r| := |I_S \backslash I_{S'}|$. Without loss of generality, we can rearrange the rows and columns of H_{I_S,I_S} such that H_{I_S,I_S} and its inverse are structured as follows:

$$H_{I_S,I_S} = \left[\begin{array}{c:c} H_{I_{S'},I_{S'}} & H_{I_{S'},I_r} \\ \hdashline H_{I_r,I_{S'}} & H_{I_r,I_r} \end{array}\right], \ (H_{I_S,I_S})^{-1} = \left[\begin{array}{c:c} A & B \\ \hdashline B^\top & C \end{array}\right], \tag{28}$$

where A is a submatrix of $(H_{I_S,I_S})^{-1}$ with the same size as $H_{I_{S'},I_{S'}}$, B has the same size as $H_{I_{S'},I_r}$, and C has the same size as H_{I_r,I_r}. Following a similar argument as in Case 1, we deduce that

$$\begin{aligned} C &= \left(H_{I_r,I_r} - H_{I_r,I_{S'}} \left(H_{I_{S'},I_{S'}}\right)^{-1} H_{I_{S'},I_r}\right)^{-1}, \\ B &= -\left(H_{I_{S'},I_{S'}}\right)^{-1} H_{I_{S'},I_r} C, \\ A &= \left(H_{I_{S'},I_{S'}}\right)^{-1} + BC^{-1}B. \end{aligned} \tag{29}$$

Given that $\left(H_{I_{S'},I_{S'}}\right)^{-1}$ has already been computed, we can efficiently calculate A, B, and C, and consequently $(H_{I_S,I_S})^{-1}$ in $O\left((t_{S'})^2 t\right) = O(td_1(d_1+d_2))$ time.

Next, we focus on computing $W^{(S)}$. Using (21), we obtain the following relationship between $W^{(S)}$ and $W^{(S')}$:

$$\begin{aligned} W^{(S)}_{I_S,:} &= \left[\begin{array}{c} W^{(S)}_{I_{S'},:} \\ \hdashline W^{(S)}_{I_r,:} \end{array}\right] = \left[\begin{array}{c:c} A & B \\ \hdashline B^\top & C \end{array}\right] \left[\begin{array}{c} G_{I_{S'},:} \\ \hdashline G_{I_r,:} \end{array}\right] = \left[\begin{array}{c} AG_{I_{S'},:} + BG_{I_r,:} \\ \hdashline B^\top G_{I_{S'},:} + CG_{I_r,:} \end{array}\right] \\ &= \left[\begin{array}{c} (H_{I_{S'},I_{S'}})^{-1} G_{I_{S'},:} + \left(BG_{I_r,:} + BC^{-1}B^\top G_{I_{S'},:}\right) \\ \hdashline B^\top G_{I_{S'},:} + CG_{I_r,:} \end{array}\right] = \left[\begin{array}{c} W^{(S')}_{I_{S'},0} \\ \hdashline \mathbf{0}_{t\times d_2} \end{array}\right] + \left[\begin{array}{c} BC^{-1} \\ \hdashline \mathbf{E}_{t\times d_2} \end{array}\right] \left(B^\top G_{I_{S'},:} + CG_{I_r,:}\right) \end{aligned} \tag{30}$$

Given that B is a $t_{S'} \times t$ matrix, C is a $t \times t$ matrix, $G_{I_{S'},:}$ is a $(t_{S'}) \times d_2$ matrix, and B, C can be computed in $O(td_1(d_1+d_2))$ time, computing $W^{(S)}$ requires $O\left((t_{S'})td_2 + td_1^2\right) = O(td_1(d_1+d_2))$ time complexity.

Finally, we compute the value of $f(S)$. Using the similar argument as (27), we obtain that

$$f(S) = f(S') + \frac{1}{2}\operatorname{Tr}\left((W^{(S)}_{I_r,:})^\top C^{-1} W^{(S)}_{I_r,:}\right). \tag{31}$$

Given that C is a $t \times t$ matrix, and $W^{(S)}_{I_r,:}$ can be computed in $O(td_1(d_1+d_2))$ time, computing $f(S)$ also requires $O(td_1(d_1+d_2))$ time complexity.

Case 3: no inclusion relationship between $I_{S'}$ and I_S.

We define $S'' = S' \cup S$, and from the definition of I_S, it follows that $I_{S''} = I_S \cap I_{S'}$. Let's denote $t_1 = |I_{S'} \backslash I_{S''}|$ and $t_2 = |I_S \backslash I_{S''}|$. The symmetry difference definition implies that $t = t_1 + t_2$.

We have already computed $f(S')$, the inverse of $H_{I_{S'},I_{S'}}$, and the optimal weight matrix $W^{(S')}$. Note that $I_{S''} \subset I_{S'}$, we can apply the results from Case 1. This allows us to compute the value of $f(S'')$, the inverse of $H_{I_{S''},I_{S''}}$, and the optimal weight matrix $W^{(S'')}$ in $O(t_1 d_1(d_1+d_2))$ time. Moreover, since $I_{S''} \subset I_S$, by applying the results from Case 2, we conclude that the value of $f(S)$, the inverse of H_{I_S,I_S}, and the optimal weight matrix $W^{(S)}$ can also be computed within $O(t_2 d_1(d_1+d_2))$ time complexity. Therefore, the overall time complexity is $O(td_1(d_1+d_2))$. This completes the proof.

A.2. Proof of Proposition 4.2

We begin by analyzing the time complexity of Algorithm 1 with general parameters t_i and p_i. To initiate the algorithm, we need to compute H^{-1} and $H^{-1}G$, which has a time complexity of $O(d_1^2(d_1+d_2))$ since H is a $d_1 \times d_1$ matrix and G is a $d_1 \times d_2$ matrix.

During iteration i of Algorithm 1, a local search is conducted by approximately solving (10) with $S' = S_{i-1}$, $\hat{t} = t_i$, and $\hat{p} = p_i$. As outlined in Section 4.2, to address (10), it suffices to calculate the impact of each element in $[p]$, as defined in (11). Assuming uniform group size Q, for any $j \in S_{i-1}$, $|I_{S_{i-1}\setminus\{j\}}\Delta I_{S_{i-1}}| = Q$, and for any $j \notin S_{i-1}$, $|I_{S_{i-1}\cup\{j\}}\Delta I_{S_{i-1}}| = Q$. Having computed $f(S_{i-1})$, the inverse of $H_{I_{S_{i-1}},I_{S_{i-1}}}$, and the optimal weight matrix for $f(S_{i-1})$, Proposition 4.1 suggests that each B_j in (11) can be computed in $O(Qd_1(d_1+d_2))$ time. Consequently, computing B_j for all $j \in [p]$ takes $O(d_1^2 d_2)$ time, given that $[d_1]$ is partitioned into p equal-sized disjoint groups $Q^1, Q^2, \ldots, Q^p$ and $pQ = d_1$.

Upon solving (10), we obtain the set S_i with $|S\Delta S'| \leq t_i$, which implies $|I_{S_i}\Delta I_{S_{i-1}}| \leq t_i Q$. Applying Proposition 4.1 again, we can calculate $f(S_i)$, the inverse of $H_{I_{S_i},I_{S_i}}$, and the optimal weight matrix for $f(S_i)$ in $O(t_i Q d_1(d_1+d_2)) = O(d_1^2(d_1+d_2))$ time complexity. Hence, the total time complexity of Algorithm 1 over T iterations is $O(Td_1^2(d_1+d_2))$.

We now focus on the specific parameter choice where $t_i = p_i$ for all $i \in [T]$. During iteration i of Algorithm 1, the process is similar to the previous case in that we still need to solve (10). However, the key difference lies in the fact that since $t_i = p_i$, as outlined in Section 4.2, we won't remove elements from S_{i-1}. Consequently, it's only necessary to compute B_j for $j \notin S_{i-1}$. This involves calculating $f(S_{i-1}\cup\{j\}) - f(S_{i-1})$ for all $i \notin S_{i-1}$. Given that $I_{S_{i-1}\cup\{j\}} \subset I_{S_{i-1}}$ and $Q = |I_{S_{i-1}}\setminus I_{S_{i-1}\cup\{j\}}|$, Lemma A.1 implies that $f(S_{i-1}\cup\{j\}) - f(S_{i-1})$ can be computed in $O(Q^2 d_1)$ time. Thus, computing B_j for all $j \notin S_{i-1}$ requires $O(pQ^2 d_1) = O(Qd_1^2)$ time.

After solving (10) and obtaining the set S_i, similarly, we can apply Proposition 4.1 to compute $f(S_i)$, the inverse of $H_{I_{S_i},I_{S_i}}$, and the optimal weight matrix for $f(S_i)$, all within $O(t_i Q d_1(d_1+d_2)$ time. The overall time complexity of Algorithm 1 over T iterations is thus

$$O\Big(\sum_{i=1}^{T} Qd_1^2 + t_i Q d_1(d_1+d_2)\Big) = O\left(\Big(\sum_{i=1}^{T} t_i\Big) Q d_1(d_1+d_2)\right) \tag{32}$$

Notably, with $t_i = p_i$ for all $i \in [T]$, we obtain $\sum_{i=1}^{T} t_i = \sum_{i=1}^{T} p_i = p' \leq \frac{d_1}{Q}$. Therefore, the overall time complexity in this case can be reduced to $O(d_1^2(d_1+d_2))$.

Finally, we turn to the memory complexity of Algorithm 1. It's important to note that, throughout the algorithm, we only need to maintain the value of $f(S)$, the inverse of H_{I_S,I_S}, and the optimal weight matrix for $f(S)$ associated with the current solution S_i. As shown in Proposition 4.1, computing these elements involves matrix multiplication with dimensions up to $d_1 \times (d_1+d_2)$. Consequently, the overall memory complexity of the algorithm is $O(d_1(d_1+d_2))$.

B. Experimental Details

B.1. Experimental setup

All experiments were carried out on a computing cluster. Experiments were run on an Intel Xeon Gold 6248 machine with 20 CPUs and a single NVIDIA V100 GPU. Our machine is equipped with 192GB of CPU RAM and 32GB of CUDA memory. We will terminate the algorithm if it leads to Out-of-Memory errors. For our experiments, we use the PyTorch library (Paszke et al., 2017) to implement all neural network models and pruning methods.

B.1.1. CNN EXPERIMENTS

Pruning settings. In our study, we focus exclusively on pruning the convolutional layers within the network. For a layer with p channels, we prune $p' = \tau p$ channels with various τ values to investigate different speedup ratios. To evaluate the effectiveness of all structured pruning methods, we utilize a calibration dataset consisting of 500 training samples.

Implementation details. Below are the configuration and implementation details for the competing methods and our framework, *OSSCAR*. It's important to note that as we assess the performance of competing methods in a one-shot setting, we shut down the fine-tuning procedure.

- *OSSCAR*: We formulate the channel pruning problem as a combinatorial problem (see (10)), and address it using

Algorithm 1. We set the number of iterations $T = p'/2$ and $t_i = p_i = 2$ for $i \in [T]$.

- MP (Mozer & Smolensky, 1989; He et al., 2018a): We implement structured pruning by calculating the Frobenius norm of weights for each channel as its magnitude and pruning p' channels with the smallest magnitude.
- CHIP (Sui et al., 2021): We utilize the authors' algorithm (codes available on GitHub) to prune channels based on channel independence.
- FPGM (He et al., 2018b): We utilize the authors' algorithm (codes available on GitHub) for channel pruning via the geometric median.
- Lasso (He et al., 2017): We utilize our own implementation, as directly applying the authors' codes to our setting is challenging. Following (He et al., 2017, Section 3), we convert the channel pruning problem into a ℓ_1-regularized problem. We solve the subproblem related to β repeatedly with increasing ℓ_1-penalty coefficients until meeting the pruning constraint, then address the subproblem related to the weight matrix W. Our implementation of this algorithm strictly adheres to the description provided by the authors in their paper.
- ThiNet (Luo et al., 2017): We utilize the authors' algorithm (codes available on GitHub) to conduct channel selection through a greedy approach, followed by a refining process to minimize reconstruction error.

Pruning strategies. Additionally, to further enhance accuracy, we implement two techniques for all *OSSCAR* and all competing methods. Firstly, as advised by (Zhang et al., 2015) and (He et al., 2017, Section 3.2), we solve the channel pruning problem layer by layer sequentially. For each layer, we minimize the squared loss between the output of the *dense* model at this layer and the output produced by the pruned weight W acting on the pruned model's input feature map. Thus, our layer-wise pruning objective for the ℓ-th layer is given as:

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} \left\| \mathrm{Conv}\left(\widehat{w}, \widehat{X}^i\right) - \mathrm{Conv}\left(w, X^i\right) \right\|^2, \tag{33}$$

Different from (1), here we replace $\mathrm{Conv}\left(\widehat{w}, X^i\right)$ by $\mathrm{Conv}\left(\widehat{w}, \widehat{X}^i\right)$, where $\widehat{X}$ denotes the dense model's input feature map, while X represents the pruned model's input feature map (with the first $\ell - 1$ layers pruned). This modification does not change the nature of the problem, and it enables pruning algorithms to more accurately approximate the output of the original model.

Secondly, for ResNet20 and ResNet50, following insights from (Luo et al., 2017, Section 3.3), we avoid pruning the first convolution layer in each residual block. This approach keeps block output dimensions unchanged and avoids extra costs in executing the residual connection. For MobileNetV1, we focus the channel pruning on the 1×1 convolutional layers as pruning the depthwise convolution layers tends to be less impactful and could significantly reduce accuracy. It's crucial to highlight that layers not directly pruned in the process are still non-dense in the pruned model. This is because some filters in these layers will become redundant when the channels they contribute to in subsequent layers are pruned. Hence, these filters can be pruned too.

B.1.2. LLM EXPERIMENTS

Pruning settings.

As discussed in Section 3, for pruning OPT models, we focus on removing attention heads and reducing the intermediate dimension in fully connected layers. We convert these two problems into MIQP problems (4), and we set $p' = \tau p$ channels with various τ values to investigate different speedup ratios. Here p is either the number of attention heads or the input dimension of the fully connected layer, depending on the sublayer we are pruning.

Implementation details. Below are the configuration and implementation details for the competing methods and our framework, *OSSCAR*.

- *OSSCAR*: We utilize Algorithm 1 to solve the combinatorial problem derived from structured pruning. For pruning attention heads, we set the number of iterations $T = p'/2$ and $t_i = p_i = 2$ for $i \in [T]$. For reducing intermediate dimensions in fully connected layers, we set the number of iterations $T = p'/10$ and $t_i = p_i = 10$ for $i \in [T]$.

- MP (Mozer & Smolensky, 1989): This method employs structured pruning by calculating the Frobenius norm of each weight group Q^i, $i = 1, 2, \ldots, p$ (as defined in (4)) as its magnitude and then pruning p' groups with the lowest magnitude.
- MP+: This approach extends MP by further refining the weights post-pruning. After performing MP, the weight matrix W is updated following (5) to reduce the approximation loss $L(W)$, where the set S is determined by MP.
- ZipLM (Kurtić et al., 2023): ZipLM's original implementation targeted models smaller than 340 million parameters. We adapt their framework for OPT models. In ZipLM's second stage, a structured SPDY search is performed, involving a 1000-step neighborhood search for various sparsity configurations across layers. Each configuration's performance is assessed on a validation set, with the best-performing configuration in terms of perplexity being selected. However, for larger OPT models, we found this SPDY search to be computationally prohibitive. For example, based on our estimation, it would take ~ 200 days for ZipLM to perform 100 search steps and validate each configuration's performance for OPT-2.7B. Consequently, we turn off the SPDY search, using on the original layer-wise sensitivity coefficients to generate the sparsity configuration.

Pruning strategies. Similar to the strategy used in CNNs, we solve the structured pruning problem layer by layer sequentially in OPT models. For each layer, our objective is to minimize the squared loss between the output of the dense model at that particular layer and the output generated by the pruned weight W acting on the pruned model's input values. It's important to note that for LLMs, this strategy is implemented only for *OSSCAR*, MP, and MP+, and not for ZipLM. In the case of ZipLM, as outlined in their paper, involves pruning each layer to different sparsity levels. This requires considering the squared loss between the output from the dense weight $\widehat{W}$ acting on the dense model's input values and the output from the pruned weight W acting on the dense model's input values.

Saving GPU memory. We employ a layer-by-layer pruning approach for *OSSCAR* and all competing methods. Consequently, we only need to load the weights of a single layer into GPU memory at any given time, offloading other weights to the CPU to conserve GPU memory. Additionally, for our calibration data – comprising 128 segments of 2048 tokens each from the C4 dataset – we store these segments on the CPU to further save GPU memory. They are loaded to the GPU individually when we need them for generating H and G in the pruning objective (3). This strategy significantly reduces GPU memory usage, allowing us to apply *OSSCAR* for pruning large OPT models with just 32GB of GPU memory.

B.2. Ablation studies and additional results

In this section, we provide additional experimental results and ablation studies focusing on one-shot structured pruning in both CNNs and LLMs.

B.2.1. Pruning performance of OPT models on PTB and C4 datasets

In this section, we present the perplexity performance of MP, MP+, ZipLM, and *OSSCAR* on the PTB (test) and C4 (validation) datasets, specifically for pruning OPT-1.3B, OPT-2.7B, and OPT-6.7B models. The results are detailed in Tables 4 and 5. Similar to the trends observed with the Wikitext dataset, *OSSCAR* demonstrates a significant reduction in perplexity compared to magnitude pruning based methods and ZipLM (state-of-the-art), particularly at higher speedup ratios.

B.2.2. Pruning performance of OPT models on zero-shot benchmarks

We evaluate the performance of OPT models pruned by MP, MP+, ZipLM, and *OSSCAR* on four zero-shot tasks: Piqa, Lambada, Arc easy, and challenge. We consider pruning OPT-1.3B and OPT-2.7B at two different speedup ratios. Tables 6 and 7 present the detailed results, demonstrating that *OSSCAR* consistently outperforms other methods by a significant margin. For instance, when pruning OPT-2.7B with a 1.7x speedup, all methods except *OSSCAR* fail to deliver meaningful results on the Lambada dataset, while OSSCAR achieves an accuracy of 38%. These findings are consistent with those from our perplexity-based evaluations, reinforcing *OSSCAR*'s efficacy in maintaining pruned models' performance.

B.2.3. Effect of calibration dataset size

In this study, we explore how varying the sample size of the calibration dataset affects the performance of one-shot structured pruning on LLMs. For all our LLM experiments previously conducted, we utilized a calibration dataset comprising $N = 128$ sequences of 2048 tokens each, sampled from C4. We now experiment with different values of N from the

Table 4: Perplexity performance on PTB for one-shot structured pruning of OPT models (1.3B, 2.7B, and 6.7B). The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods we take ten runs and report the mean and standard error.

Model	Speedup	MP	MP+	ZipLM	*OSSCAR*
OPT-1.3B	1.2x	164.8 (±0.06)	24.54 (±0.08)	**18.57** (±0.21)	19.79 (±0.14)
	1.3x	1166 (±0.72)	48.60 (±0.83)	63.61 (±2.63)	**24.45** (±0.23)
	1.4x	7147 (±20.8)	795.0 (±53.1)	477.9 (±62.0)	**30.08** (±0.42)
	1.7x	7762 (±31.8)	1646 (±90.8)	1786 (±161)	**40.74** (±1.39)
	2.0x	6796 (±8.87)	4320 (±224)	3716 (±386)	**62.02** (±2.94)
	2.6x	8092 (±5.89)	5628 (±124)	5330 (±312)	**97.92** (±4.17)
	3.3x	9612 (±30.8)	6071 (±127)	9616 (±692)	**170.9** (±9.65)
OPT-2.7B	1.2x	365.9 (±1.08)	24.90 (±0.09)	**15.15** (±0.03)	17.21 (±0.20)
	1.3x	3281 (±123)	38.64 (±0.23)	26.91 (±2.29)	**19.80** (±0.30)
	1.4x	10062 (±969)	88.62 (±1.14)	257.5 (±58.4)	**24.91** (±0.41)
	1.7x	7053 (±52.7)	1380 (±152)	1289 (±176)	**31.20** (±0.79)
	2.0x	8530 (±29.8)	4803 (±248)	3267 (±374)	**43.47** (±1.11)
	2.4x	6854 (±16.1)	7407 (±166)	8017 (±852)	**70.95** (±1.81)
	3.0x	10166 (±37.9)	7780 (±171)	12624 (±968)	**122.7** (±3.08)
OPT-6.7B	1.1x	208.2 (±0.40)	22.45 (±0.10)	OOM	**17.21** (±0.44)
	1.2x	4086 (±37.8)	53.57 (±1.64)		**18.96** (±0.38)
	1.5x	15915 (±186)	1056 (±145)		**22.24** (±0.58)
	1.7x	15668 (±110)	2858 (±162)		**26.78** (±0.57)
	2.0x	11537 (±71.3)	5833 (±90.1)		**34.80** (±1.10)
	2.6x	12524 (±69.0)	7142 (±177)		**54.70** (±1.20)
	3.3x	13585 (±87.9)	6815 (±220)		**102.4** (±3.57)

Table 5: Perplexity performance on C4 for one-shot structured pruning of OPT models (1.3B, 2.7B, and 6.7B). The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods we take ten runs and report the mean and standard error.

Model	Speedup	MP	MP+	ZipLM	*OSSCAR*
OPT-1.3B	1.2x	84.29 (±0.03)	18.32 (±0.04)	**14.59** (±0.03)	15.58 (±0.04)
	1.3x	814.6 (±0.80)	26.24 (±0.21)	41.81 (±1.57)	**16.78** (±0.09)
	1.4x	5630 (±9.11)	322.1 (±15.4)	332.6 (±37.7)	**18.73** (±0.17)
	1.7x	6188 (±24.6)	847.5 (±74.7)	943.4 (±39.5)	**22.02** (±0.25)
	2.0x	6327 (±8.17)	2994 (±46.9)	1823 (±103)	**28.60** (±0.42)
	2.6x	8046 (±9.65)	4264 (±92.3)	2693 (±100)	**46.29** (±0.78)
	3.3x	9611 (±21.9)	5365 (±99.5)	5643 (±425)	**84.49** (±1.97)
OPT-2.7B	1.2x	296.0 (±0.73)	19.62 (±0.10)	**12.63** (±0.00)	13.85 (±0.04)
	1.3x	4108 (±164)	27.94 (±0.30)	17.61 (±0.73)	**14.88** (±0.08)
	1.4x	11538 (±1386)	46.83 (±0.85)	189.3 (±37.4)	**16.47** (±0.13)
	1.7x	10179 (±87.7)	821.7 (±128)	712.8 (±103)	**19.17** (±0.23)
	2.0x	10345 (±21.5)	4298 (±213)	1711 (±195)	**24.25** (±0.34)
	2.4x	9363 (±17.3)	6944 (±167)	3660 (±331)	**36.91** (±0.78)
	3.0x	14822 (±40.7)	7329 (±111)	6567 (±599)	**65.42** (±1.17)
OPT-6.7B	1.1x	226.3 (±0.04)	16.42 (±0.10)	OOM	**12.26** (±0.04)
	1.2x	3694 (±4.38)	24.89 (±0.41)		**13.16** (±0.10)
	1.5x	19913 (±142)	271.5 (±16.9)		**14.48** (±0.17)
	1.7x	17843 (±19.5)	1665 (±122)		**16.56** (±0.30)
	2.0x	16617 (±51.8)	5834 (±264)		**20.31** (±0.43)
	2.6x	18948 (±41.6)	7744 (±203)		**29.55** (±0.17)
	3.3x	20887 (±56.0)	8000 (±126)		**49.70** (±1.41)

set $\{128, 256, 512, 1024, 2048, 4096\}$ to assess its impact on the perplexity of pruned models. The results of applying *OSSCAR* to prune OPT models ranging from 1.3B to 30B for a 2x speedup with these varied sample sizes N are shown in Table 8. It is observed that for a 2x speedup, the performance improves by up to ~ 3.5 perplexity units when increasing the sample size N from 128 to 2048. Beyond this point, the performance plateaus, showing no significant change with further increases in sample size.

As the number of samples increases, the total time required for applying *OSSCAR* to prune the model also rises. The pruning process consists of two parts: constructing matrices H and G as defined in (3) for each layer, and solving the structured

Table 6: Performance on zero-shot tasks (larger values indicate better performance) for one-shot structured pruning of OPT-1.3B. The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods we take ten runs and report the mean.

Speedup	Method	Piqa	Lambada	Arc easy	Arc challenge
1x	Dense	71.70	53.30	56.99	23.46
1.3x	MP	55.44	0.0	32.45	23.46
	MP+	66.70	26.72	48.53	21.84
	ZipLM	61.92	37.34	40.11	20.05
	OSSCAR	**71.22**	**45.88**	**56.44**	**23.72**
1.7x	MP	51.25	0.0	25.80	20.14
	MP+	53.64	0.0	28.66	21.50
	ZipLM	55.00	0.5	28.37	21.42
	OSSCAR	**68.23**	**31.98**	**51.81**	**23.12**

Table 7: Performance on zero-shot tasks (larger values indicate better performance) for one-shot structured pruning of OPT-2.7B. The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods we take ten runs and report the mean.

Speedup	Method	Piqa	Lambada	Arc easy	Arc challenge
1x	Dense	73.83	57.67	60.77	26.88
1.3x	MP	59.25	0.0	37.92	23.72
	MP+	67.74	26.14	49.28	24.23
	ZipLM	52.03	50.71	27.31	20.99
	OSSCAR	**73.50**	**54.73**	**59.72**	**25.94**
1.7x	MP	54.13	0.0	27.15	20.65
	MP+	55.77	0.27	31.14	19.03
	ZipLM	52.50	0.55	25.97	21.50
	OSSCAR	**71.49**	**38.67**	**53.79**	**23.72**

Table 8: Perplexity performance on Wikitext for applying one-shot structured pruning of OPT models (1.3B, 2.7B, 6.7B, 13B and 30B), with varying sizes of calibration data. All pruned models shown in this table have a 2x improvement in inference time compared to dense models. For all models and calibration data sizes, we take ten runs and report the mean and standard error.

Number of Samples	OPT-1.3B	OPT-2.7B	OPT-6.7B	OPT-13B	OPT-30B
128	38.16 ($\pm$0.52)	29.49 ($\pm$0.53)	25.03 ($\pm$0.73)	22.59 ($\pm$0.59)	17.11 ($\pm$0.30)
256	37.29 ($\pm$0.53)	28.29 ($\pm$0.36)	23.92 ($\pm$0.21)	22.34 ($\pm$0.34)	16.45 ($\pm$0.17)
512	36.04 ($\pm$0.25)	27.91 ($\pm$0.25)	23.25 ($\pm$0.25)	22.27 ($\pm$0.35)	16.07 ($\pm$0.11)
1024	35.91 ($\pm$0.09)	27.73 ($\pm$0.34)	23.76 ($\pm$0.28)	21.79 ($\pm$0.29)	15.86 ($\pm$0.06)
2048	**34.87** ($\pm$0.11)	**27.09** ($\pm$0.12)	23.15 ($\pm$0.15)	21.34 ($\pm$0.10)	**15.77** ($\pm$0.04)
4096	34.97 ($\pm$0.11)	27.38 ($\pm$0.30)	**22.91** ($\pm$0.17)	**20.95** ($\pm$0.17)	OOM*

*OOM denotes out of CPU memory here.

pruning problem using Algorithm 1. The time taken to construct matrices H and G grows linearly with the number of samples. However, the time required to apply Algorithm 1 is largely unaffected by the number of samples. This is because the dimensions of H and G (d_1 and d_2) do not depend on N; hence, as per Proposition 4.2, the time complexity of the algorithm remains the same. For instance, when pruning the OPT-13B model with $N = 128$ samples, constructing H and G takes 20 minutes, and Algorithm 1 takes 25 minutes; for $N = 2048$ samples, constructing H and G takes 320 minutes, while Algorithm 1 takes 28 minutes.

Table 9: Test accuracy for applying *OSSCAR* to prune CNNs (ResNet20, MobileNetV1 and ResNet50), under different choices of parameters $\hat{t} = \hat{p}$. For all models and parameters, we take ten runs and report the mean and standard error.

Speedup	$\hat{t} = \hat{p}$	Models		
		ResNet20	MobileNetV1	ResNet50
1.4x	1	87.69 (±0.14)	**68.89** (±0.09)	65.29 (±0.29)
	2	**87.70** (±0.16)	68.89 (±0.08)	**65.32** (±0.27)
	5	87.60 (±0.13)	68.88 (±0.10)	65.24 (±0.24)
	10	87.51 (±0.11)	68.79 (±0.09)	65.23 (±0.23)
1.9-2.0x	1	**81.34** (±0.50)	**58.42** (±0.41)	**38.56** (±0.64)
	2	81.10 (±0.49)	58.39 (±0.36)	38.45 (±0.63)
	5	80.89 (±0.42)	58.32 (±0.30)	38.44 (±0.64)
	10	80.56 (±0.52)	58.20 (±0.41)	38.23 (±0.36)

Table 10: Perplexity performance on Wikitext for applying *OSSCAR* to prune OPT models (1.3B, 2.7B and 6.7B), under different choices of parameters $\hat{t} = \hat{p}$. For all models and parameters, we take ten runs and report the mean and standard error.

Speedup	$\hat{t} = \hat{p}$	Models		
		OPT-1.3B	OPT-2.7B	OPT-6.7B
1.4x-1.5x	1	20.53 (±0.35)	17.23 (±0.19)	14.94 (±0.19)
	5	20.54 (±0.38)	**17.20** (±0.20)	14.93 (±0.20)
	10	**20.50** (±0.36)	17.23 (±0.20)	14.93 (±0.18)
	50	20.60 (±0.39)	17.26 (±0.19)	**14.84** (±0.20)
	100	20.65 (±0.37)	17.28 (±0.20)	14.89 (±0.20)
	500	20.65 (±0.37)	17.54 (±0.22)	15.29 (±0.21)
2.0x	1	38.04 (±0.59)	29.58 (±0.47)	24.92 (±0.79)
	5	38.08 (±0.55)	29.64 (±0.50)	25.00 (±0.76)
	10	38.16 (±0.52)	**29.49** (±0.53)	25.02 (±0.73)
	50	38.08 (±0.52)	29.63 (±0.52)	25.37 (±0.63)
	100	**38.03** (±0.51)	29.62 (±0.58)	25.43 (±0.63)
	500	38.65 (±0.60)	30.04 (±0.70)	25.76 (±0.63)

Table 11: Pruning time for applying *OSSCAR* to prune OPT models (1.3B, 2.7B and 6.7B), under different choices of parameters $\hat{t} = \hat{p}$. For all models and parameters, we take ten runs and report the mean and standard error.

Speedup	$\hat{t} = \hat{p}$	Models		
		OPT-1.3B	OPT-2.7B	OPT-6.7B
1.4x-1.5x	1	376 (±8.0)	724 (±5.0)	2592 (±10)
	5	244 (±21)	434 (±3.0)	1140 (±6.0)
	10	201 (±6.0)	384 (±3.0)	974 (±4.0)
	50	185 (±4.0)	353 (±5.0)	825 (±5.0)
	100	185 (±3.0)	348 (±4.0)	805 (±6.0)
	500	183 (±5.0)	347 (±6.0)	802 (±6.0)
2.0x	1	526 (±8.0)	1108 (±6.0)	4009 (±13)
	5	256 (±9.0)	500 (±3.0)	1419 (±4.0)
	10	218 (±6.0)	418 (±2.0)	1118 (±15)
	50	189 (±4.0)	366 (±4.0)	848 (±2.0)
	100	189 (±7.0)	361 (±2.0)	821 (±3.0)
	500	189 (±3.0)	369 (±11)	798 (±3.0)

B.2.4. Performance of *OSSCAR* after retraining

In this work, we focus on one-shot pruning due to its significantly lower memory and computational demands. This enables *OSSCAR* to scale to models with up to 30B parameters on a single GPU. In contrast, previous studies (Chen et al., 2023; Benbaki et al., 2023) have shown that while gradual pruning/retraining can further improve model performance, it requires substantial training time and appropriate parameter choices. For example, *OSSCAR* can prune a Resnet50 model in less than 10 minutes, whereas retraining can take several days.

Although *OSSCAR* is a one-shot pruning framework, the models it prunes can be efficiently retrained to achieve high accuracy. To demonstrate this, we pruned Resnet20 models using *OSSCAR* and compared them with models pruned by MP, CHIP, FPGM, Lasso, and ThiNet on the Cifar10 dataset. All models were retrained with a learning rate of 1e-4, momentum of 0.9, weight decay of 5e-5, and for 10 epochs. The accuracy of the retrained models is shown in Table 12. While fine-tuning narrows the performance gap between pruned models, *OSSCAR* still outperforms the others after retraining. This is likely because *OSSCAR* achieves higher accuracy during the initial pruning stage, as illustrated in Section 5.1. Moreover, this suggests that models pruned by other methods require more retraining epochs and parameter tuning to reach the same performance as those pruned by *OSSCAR*. Therefore, if sufficient computational resources are available for retraining after one-shot pruning, *OSSCAR* is the preferred choice because it requires fewer resources to retrain pruned models to achieve high accuracy.

B.2.5. Performance of Algorithm 1 across various parameter settings

In this section, we explore how parameters $\hat{t}$ and $\hat{p}$ in (10) influence the performance of Algorithm 1.

We begin by setting $\hat{t} = \hat{p}$ to assess the impact of $\hat{t}$ on both the solution quality and the time complexity of the algorithm. Specifically, we set $t_i = p_i = \hat{t}$ for all iterations $i \in [T]$ and set the number of iterations $T = p'/\hat{t}$ for Algorithm 1

Table 12: Accuracy performance of retrained ResNet20 models on Cifar10, pruned using various structured pruning methods to achieve different speedup ratios. Each model was retrained with a learning rate of 1e-4, momentum of 0.9, weight decay of 5e-5, for 10 epochs. The speedup ratio denotes the inference time improvement of pruned models over dense models. For all methods, we take ten runs and report the mean.

Speedup	MP	CHIP	FPGM	Lasso	ThiNet	OSSCAR
1.3x	89.42	87.64	88.55	**89.76**	89.66	89.59
1.4x	87.46	86.05	86.82	88.42	88.72	**88.80**
1.7x	85.53	83.37	85.28	86.99	86.83	**87.12**
2.0x	83.35	79.21	82.39	84.82	84.59	**85.63**
2.6x	78.60	73.93	77.50	79.34	79.44	**80.46**
3.5x	69.78	65.14	69.41	73.06	72.81	**74.54**

We detail in Table 9 the accuracy performance of *OSSCAR* for pruning CNNs with different $\hat{t}$ values. For OPT models, we display the perplexity performance and pruning time in Tables 10 and 11, respectively. Notably, when pruning attention heads in OPT models, we consistently set $\hat{t} = 2$—in this context, Algorithm 1 typically converges in less than 50 iterations regardless of parameter choice, and varying $\hat{t}$ offers no advantage. Different $\hat{t}$ values are considered when reducing the intermediate dimension in fully connected layers.

We observe that smaller $\hat{t}$ values yield marginally better performance, although the difference is not statistically significant. This suggests that *OSSCAR*'s accuracy/perplexity performance exhibits low sensitivity to the choice of $\hat{t}$. In contrast, the pruning time can be significantly reduced (2-5x) when increasing $\hat{t}$ from 1 to 100 in the pruning of OPT models. While Proposition 4.2 suggests that the theoretical complexity of Algorithm 1 does not depend on $\hat{t}$ when $\hat{t} = \hat{p}$, we find that larger $\hat{t}$ values can expedite the algorithm in practice, due to benefits from vectorization. Consequently, for OPT models, practical choices of $\hat{k} = \hat{t} = 10$ or 100 can offer an effective balance of performance and faster pruning times.

Next, we examine the impact of selecting a small $\hat{p}$ on the performance of our algorithm. In Algorithm 1, the sum of p_i chosen in each iteration needs to be a fixed number. Thus, choosing for a small $\hat{p}$ for all iterations leads to a substantially larger number of iterations, potentially degrading performance. To address this, we consider the following two parameter settings:

1. We set the number of iterations $T = p'/\hat{t}$ and set $t_i = p_i = \hat{t}$ for all iterations $i \in [T]$.
2. We set the number of iterations $T = p'/\hat{t} + 30$. For the first $p'/\hat{t}$ iterations, we set $t_i = p_i = \hat{t}$, and for the final 30 iterations, we set $t_i = \hat{t}$ and $p_i = 0$.

The first setting aligns with the parameters used in all our previous experiments. In the second setting, we initially follow the parameters of the first case, then switch to $\hat{p} = 0$ for the last 30 iterations to implement a local swapping strategy. This involves exchanging elements within the set S with more impactful ones from outside S to achieve a set with a lower objective value, as depicted in the right part of Figure 4. We refer to the first setting as "nested local search" and the second as "non-nested local search".

We assess the above two parameter settings in our ablation study. For ResNet20 and MobileNetV1, we set $\hat{t} = 5$, and for ResNet50, $\hat{t} = 10$. In the case of OPT models, we use $\hat{t} = 2$ for pruning attention heads and $\hat{t} = 100$ for reducing the intermediate dimension in fully connected layers. The accuracy results for pruning CNNs are presented in Table 13, and the perplexity results for pruning OPT models are presented in Table 14.

We observe that *OSSCAR* with non-nested local search approach tends to yield better results in most scenarios, though the improvements are not always statistically significant. This finding suggests that in Algorithm 1, we may use a small $\hat{p}$ for non-nested local search to further refine the quality of solutions at the expense of more pruning time.

Table 13: Test accuracy for applying *OSSCAR* to prune CNNs (ResNet20, MobileNetV1 and ResNet50), with nested/non-nested local search. For all models and parameters, we take ten runs and report the mean and standard error.

Model	Speedup	*OSSCAR* with *nested* local search	*OSSCAR* with *non-nested* local search
ResNet20	1.3x	**89.28** (±0.14)	89.20 (±0.10)
	1.4x	**87.60** (±0.13)	87.55 (±0.21)
	1.7x	**84.98** (±0.22)	84.81 (±0.19)
	2.0x	80.89 (±0.42)	**81.05** (±0.48)
	2.6x	72.06 (±0.95)	**72.09** (±0.81)
	3.5x	60.03 (±1.30)	**60.97** (±0.96)
MobileNetV1	1.3x	70.66 (±0.08)	**70.68** (±0.08)
	1.4x	68.88 (±0.10)	**68.89** (±0.10)
	1.5x	66.36 (±0.16)	**66.40** (±0.09)
	1.7x	62.98 (±0.22)	**63.05** (±0.25)
	1.9x	58.32 (±0.30)	**58.48** (±0.39)
	2.2x	52.03 (±0.53)	**52.06** (±0.52)
ResNet50	1.2x	74.34 (±0.09)	**74.37** (±0.06)
	1.3x	69.18 (±0.15)	**69.20** (±0.20)
	1.4x	**65.23** (±0.23)	65.21 (±0.24)
	1.6x	58.95 (±0.31)	**59.05** (±0.44)
	1.7x	50.07 (±0.38)	**50.10** (±0.59)
	1.9x	38.23 (±0.36)	**38.25** (±0.52)

Table 14: Perplexity performance on Wikitext for applying *OSSCAR* to prune OPT models (1.3B, 2.7B and 6.7B), with nested/non-nested local search. For all models and parameters, we take ten runs and report the mean and standard error.

Model	Speedup	*OSSCAR* with *nested* local search	*OSSCAR* with *non-nested* local search
OPT-1.3B	1.2x	15.50 (±0.09)	**15.38** (±0.06)
	1.3x	17.40 (± 0.11)	**17.30** (±0.10)
	1.4x	20.76 (±0.20)	**20.62** (±0.18)
	1.7x	26.83 (±0.32)	**26.55** (±0.35)
	2.0x	39.96 (±0.65)	**39.38** (±0.60)
	2.6x	76.51 (±1.97)	**73.66** (±1.86)
	3.3x	160.9 (±6.12)	**156.7** (±4.57)
OPT-2.7B	1.2x	13.30 (±0.08)	**13.17** (±0.07)
	1.3x	15.17 (±0.13)	**15.05** (±0.13)
	1.4x	17.43 (±0.14)	**17.33** (±0.16)
	1.7x	21.88 (±0.25)	**21.68** (±0.26)
	2.0x	30.58 (±0.33)	**30.24** (±0.33)
	2.4x	54.90 (±1.04)	**53.11** (±0.82)
	3.0x	107.0 (±2.35)	**105.3** (±2.77)
OPT-6.7B	1.1x	11.55 (±0.03)	**11.36** (±0.03)
	1.2x	12.98 (±0.08)	**12.79** (±0.08)
	1.5x	15.00 (±0.10)	**14.94** (±0.13)
	1.7x	18.97 (±0.23)	**18.92** (±0.22)
	2.0x	26.01 (±0.36)	**26.05** (±0.36)
	2.6x	47.55 (±0.72)	**46.63** (±0.76)
	3.3x	106.9 (±3.88)	**100.8** (±3.01)