FALCON: FLOP-Aware Combinatorial Optimization for Neural Network Pruning

Xiang Meng MIT
Wenyu Chen MIT
Riade Benbaki MIT
Rahul Mazumder MIT

Abstract

The increasing computational demands of modern neural networks present deployment challenges on resource-constrained devices. Network pruning offers a solution to reduce model size and computational cost while maintaining performance. However, most current pruning methods focus primarily on improving sparsity by reducing the number of nonzero parameters, often neglecting other deployment costs such as inference time, which are closely related to the number of floating-point operations (FLOPs). In this paper, we propose `FALCON`, a novel combinatorial-optimization-based framework for network pruning that jointly takes into account model accuracy (fidelity), FLOPs, and sparsity constraints. A main building block of our approach is an integer linear program (ILP) that simultaneously handles FLOP and sparsity constraints. We present a novel algorithm to approximately solve the ILP. We propose a novel first-order method for our optimization framework which makes use of our ILP solver. Using problem structure (e.g., the low-rank structure of approx. Hessian), we can address instances with millions of parameters. Our experiments demonstrate that `FALCON` achieves superior accuracy compared to other pruning approaches within a fixed FLOP budget. For instance, for ResNet50 with 20% of the total FLOPs retained, our approach improves the accuracy by 48% relative to state-of-the-art. Furthermore, in gradual pruning settings with re-training between pruning steps, our framework outperforms existing pruning methods, emphasizing the significance of incorporating both FLOP and sparsity constraints for effective network pruning.

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

1 INTRODUCTION

The remarkable success of modern neural networks has been accompanied by a surge in computational requirements (Devlin et al., 2018; Brown et al., 2020), which pose significant challenges in deploying these models on resource-constrained devices such as mobile phones and Internet-of-things (IoT) devices. Network pruning is emerging as a promising framework towards mitigating computational burdens. Pruning removes redundant or less important parameters, with the goal of retaining high performance while significantly reducing model size and computational complexity (Blalock et al., 2020).

Common approaches for network pruning include (i) impact-based (LeCun et al., 1989; Hassibi and Stork, 1992; Dong et al., 2017; Singh and Alistarh, 2020) and (ii) optimization-based methods (Yu et al., 2022; Benbaki et al., 2023). Impact-based techniques eliminate weights based on the degree to which the removal of each individual weight might influence the loss function. Existing works mostly employ clever heuristics for evaluating impacts based on factors such as absolute values (also known as magnitude pruning, (Hanson and Pratt, 1988)), change in the loss function (LeCun et al., 1989; Hassibi and Stork, 1992), or network connection sensitivity (Lee et al., 2018). Despite their intuitive appeal, impact-based pruning may fall short of capturing the *joint* effect of simultaneously removing multiple weights, resulting in suboptimal pruning outcomes. Yu et al. (2022) present optimization-based approaches for network pruning that consider the combined effect of pruning multiple weights at a time. Subsequently, CHITA (Benbaki et al., 2023) formulate sparse pruning as an ℓ_0-regularized sparse regression problem (Hazimeh and Mazumder, 2020) and propose new optimization algorithms which are memory-efficient, and scalable, and result in state-of-the-art accuracy-sparsity tradeoffs on various examples.

The approaches mentioned above aim to sparsify the network by reducing the number of non-zero (NNZ) weights, thus lowering the memory storage of the deployed model. However, these approaches do not directly consider other deployment costs or their proxies. For example, inference time and energy consumption are two important considerations

arising in practical deployment scenarios, and the number of floating-point operations (FLOPs) has been proposed as a reasonable proxy for those two factors (Yang et al., 2017). While there is a sizable amount of work on network pruning to reduce NNZ, only a few studies attempt to directly reduce or control for the number of FLOPs during pruning. Veniat and Denoyer (2018); Tang et al. (2018) propose dense-to-sparse training algorithms that incorporate FLOPs minimization by adding a weighted ℓ_0-norm of weights to the optimization objective. They introduced stochastic gates to selectively mask weights, making the weighted ℓ_0-regularized objective differentiable under certain distributions. Singh and Alistarh (2020) design an impact-based pruning method using the empirical Fisher approximation of the Hessian matrix. They propose a FLOP-aware heuristic that considers the FLOPs for each parameter while pruning. While Kusupati et al. (2020) does not directly consider FLOPs as part of the objective, they achieve state-of-the-art accuracy-vs-FLOPs trade-offs by learning relatively uniform NNZ budgets across layers (Singh and Alistarh, 2020, Appendix S6).

In this paper, we introduce FALCON (FLOP-Aware ℓ_0-based Combinatorial Optimization for Network pruning), an efficient optimization-based framework for network pruning that considers both sparsity and FLOP constraints. Our optimization formulation allows us to directly adjust FLOP and NNZ budgets, resulting in pruned networks with a good accuracy-FLOPs-sparsity tradeoffs. By effectively controlling for inference time-and-memory usage via their proxies NNZ-and-FLOPs, our approach can potentially allow practitioners to set and achieve their desired compression targets while retaining accuracy as much as possible.

Magnitude pruning (MP) is a simple and popular method for pruning weights to reduce NNZ. We first generalize the notion of MP to accommodate NNZ and FLOP budgets simultaneously. We demonstrate that this generalized MP framework can be formulated as an integer linear program (ILP), and we develop an efficient algorithm with a linear convergence rate to obtain high-quality solutions for the problem. This provides insights into how the two constraints (NNZ and FLOP) interact with each other, thereby enhancing our understanding of the pruning problem.

Obtaining a good solution to the ILP is an important component of FALCON. While parts of our framework (e.g., the local quadratic approximation based on Hessian) FALCON draw inspiration from recent studies (Singh and Alistarh, 2020; Yu et al., 2022; Benbaki et al., 2023), there are new contributions in this work. Current methods that address a sparsity constraint would not directly extend to handle the additional FLOP constraint. Therefore, to simultaneously address both FLOP and sparsity constraints, we explore new optimization techniques. In particular, we propose a discrete first-order (DFO) algorithm: in each iteration, we consider an ILP similar to the one arising in the generalized MP problem mentioned above. In addition, we leverage the low-rank structure of the approximated Hessian matrix for efficient optimization, bypassing the need for a costly Hessian computation and storage.

Our experiments reveal that, given a fixed FLOP budget, our pruned models exhibit significantly better accuracy compared to other pruning approaches. We also conduct several ablation studies to highlight the importance of introducing joint budget constraints, as opposed to mere FLOP minimization. Moreover, when employed in a gradual pruning setting (Gale et al., 2019; Blalock et al., 2020), where retraining between pruning steps is performed, our pruning framework results in substantial performance gains compared to state-of-the-art pruning methods.

Contributions We summarize our contributions as follows:

- We introduce FALCON, a novel optimization-based framework for network pruning that accounts for both NNZ and FLOP budgets. Our approach achieves an optimal balance between NNZ and FLOP—proxies for inference time and memory usage—while preserving as much as possible the accuracy of highly compressed networks. Our novel Discrete First-Order (DFO) algorithm iteratively solves an ILP with both FLOP and NNZ constraints. By exploiting problem-structure, we can efficiently prune the network for large problem-sizes.
- We generalize the MP method to handle designated sparsity and FLOP constraints by formulating it as an ILP. We develop an efficient algorithm for solving the relaxed linear program (LP) and demonstrate that high-quality integer solutions can be recovered from LP solutions. This generalized MP framework is an important tool for FLOP-aware pruning, and plays a critical role in the DFO method of FALCON.
- Our numerical results showcase FALCON's superior accuracy compared to other pruning approaches within a fixed FLOP budget. Notably, without retraining, FALCON prunes a ResNet50 network to just 1.2 billion FLOPs (30% of total FLOPs) with 73% test accuracy, a mere 4% reduction compared to the dense model, significantly outperforming state-of-the-art results (61%). Moreover, in gradual pruning settings with re-training between pruning steps, our framework surpasses existing pruning methods. Our code is publicly available at: https://github.com/mazumder-lab/FALCON.

This paper follows prior research on algorithms for unstructured network pruning. Our main focus is on proposing an optimization method that can handle the challenging task of unstructured pruning under both FLOP and NNZ constraints, useful proxies for inference time and memory requirements. Networks pruned via our novel framework can be deployed on specialized hardware, such as the efficient inference engine (Han et al., 2016), to achieve noticeable enhancements in inference time. On a related note, a different line of work,

known as structured pruning (see, e.g., He et al. (2017); Guo et al. (2020)), seeks to remove entire components of the network, such as channels and neurons. This line of work is more readily suited to efficient practical hardware utilization but can potentially lead to greater accuracy loss for the same reduction in model size. The focus of our work here is on unstructured pruning.

2 PROBLEM FORMULATION

We consider a neural network with a loss function defined as $\mathcal{L}(w) = (1/N)\sum_{i=1}^{N} \ell_i(w)$, where $w \in \mathbb{R}^p$ are the weights of the network, N denotes the number of data points or samples, and $\ell_i(w)$ is a twice-differentiable function for the i-th sample.

A weighted ℓ_0 formulation of FLOPs The FLOPs of a neural network is the total number of floating point operations required for a single forward pass. We denote the FLOP cost for a parameter as the number of floating point operations associated with the parameter during a forward pass. For instance, the FLOPs of a scalar value within a kernel in a convolution layer can be calculated as FLOPs = (Output height) $\times$ (Output width).

In our setting, we introduce the vector $f = (f_1, \ldots, f_p)$, where the i-th element, f_i, represents the FLOP cost for the i-th parameter in the network. We focus on the theoretical FLOP cost of the pruned network, which can be expressed as $\|w\|_{0,f} := \sum_{i=1}^{p} f_i \mathbf{1}_{w_i \neq 0}$. Here, $\mathbf{1}_{w_i \neq 0}$ denotes the indicator function with a value of 1 if w_i is not pruned and 0 otherwise. This formulation appears in earlier work such as Singh and Alistarh (2020); Kusupati et al. (2020).

The structure of FLOP cost It is important to recognize that in most neural network architectures, including ResNet (He et al., 2016) and MobileNet (Howard et al., 2017) considered in our numerical experiments, all parameters within the same layer contribute equally to the total FLOP cost. We formalize this observation as the following fact:

Fact 2.1. *All parameters in the network can be divided into L disjoint groups $C_1, C_2, \ldots, C_L$ such that for any $j \in [L]$, all parameters within group C_j has the same FLOP cost f^j.*

We can divide the parameters into groups based on the layer they belong to, and the number of groups L is usually much smaller than p ($L \ll p$). Furthermore, we may improve this partition, as parameters across different layers may also share the same FLOP cost. This is an important observation that we will make use of both in our theoretical results and in our implementation.

Pruning goal Our goal is to minimize the number of non-zero parameters (NNZ) in the network and the FLOP cost during inference while preserving its performance as much as possible. Specifically, we are given a pre-trained weight vector $\bar{w} \in \mathbb{R}^p$, an NNZ budget S, and a FLOP budget F. We seek to compute a new weight vector $w \in \mathbb{R}^p$ that satisifes the following desiderata[1]:

- **Retain model performance**: The loss function at w should be as close as possible to the loss before pruning: $\mathcal{L}(w) \approx \mathcal{L}(\bar{w})$.
- **FLOP budget**: The FLOP cost must be limited by the constraint $\|w\|_{0,f} = \sum_{i=1}^{p} f_i \mathbf{1}_{w_i \neq 0} \leq F$.
- **Sparsity constraint**: The number of non-zero weights in w must adhere to the NNZ budget: $\|w\|_0 := \sum_{i=1}^{p} \mathbf{1}_{w_i \neq 0} \leq S$.

Throughout the paper, given a weight vector $w \in \mathbb{R}^p$, we denote its sparsity by $1 - \|w\|_0/p$, and its NNZ by $\|w\|_0$. We refer to S as the NNZ budget. We interchangeably use the terms sparsity constraint, cardinality constraint and budget constraint (on NNZ) to refer to the constraint $\|w\|_0 \leq S$.

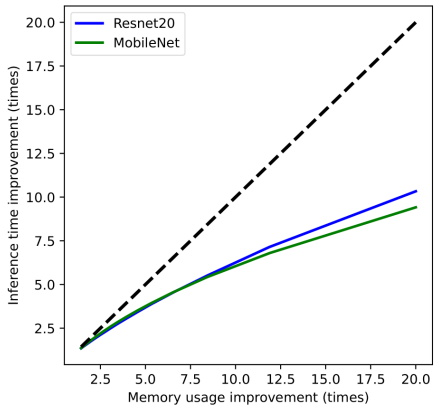


Figure 1: Inference time improvement (measured by FLOPs) vs. memory usage improvement (measured by NNZ) of CHITA (Benbaki et al., 2023). Dashed black line denotes equal improvements.

A motivating example To motivate the use of the FLOP constraint, in Figure 1, we investigate the improvements of both inference time (measured by FLOPs) and memory usage (measured by NNZ) given by the pruned model. The pruning is done by a state-of-the-art sparse pruning method CHITA which aims to reduce the NNZ of the model. Figure 1 shows that the model usually leads to much less improvement in terms of inference time (FLOPs) than in memory (NNZ). In particular, although a highly pruned model can lead to a 20$\times$ improvement in memory, its actual inference time improvement is only 8-10$\times$. This suggests that NNZ alone is not a good network pruning metric for inference time improvement—this motivates us to incorporate the FLOP constraint to accompany the commonly-used sparsity constraint in the existing literature. Furthermore, as evidenced in Section 5.1.2, combining sparsity and FLOP

[1]We assume, without loss of generality, that $\{f_i\}_{i=1}^{p}$ are not identical. This differentiates the FLOP budget from the sparsity constraint.

constraints allows for a relatively uniform sparsity across layers of the network, leading to improved accuracy.

3 ILP-BASED MAGNITUDE PRUNING UNDER NNZ AND FLOP BUDGETS

Magnitude pruning, a popular method in the neural network pruning literature, takes a simple form when addressing the sparsity constraint alone. In particular, in the context of unstructured sparsity, this eliminates a portion of parameters with the smallest absolute values while retaining the remaining weights. However, the pruning process becomes more complex when we consider both FLOP and NNZ constraints. To see this, we will first consider a magnitude-based pruning framework incorporating both constraints — this sheds insights into our proposed optimization formulation of network pruning with both FLOP and sparsity constraints (cf Section 4).

3.1 Integer Linear Programming (ILP) Formulation

The MP approach for sparse pruning is to select a set of parameters with the least absolute values to remove while adhering to cardinality constraints. For each $i \in [p]$, let the binary variable $z_i = \mathbf{1}_{w_i \neq 0}$ denote if weight w_i is retained or pruned, and $I_i = \bar{w}_i^2$ denote the squared magnitude of the i-th parameter in the network. Then, the MP method is equivalent to solving the following cardinality-constrained optimization problem:

$$\max_{z \in \{0,1\}^p} \quad Q_I(z) := \sum_{i=1}^{p} I_i z_i, \ \text{ s.t. } \ \sum_{i=1}^{p} z_i \leq S, \qquad (1)$$

which can be readily solved via sorting (see Appendix A.3): After Problem (1) is solved for z_i's, the pruned weights obtained by MP are given as $w_i = \bar{w}_i z_i$ for all i.

Here, we generalize the MP approach in Problem (1) to include an additional FLOP constraint in the model to cater to both FLOP and NNZ budget constraints. Note that the number of theoretical FLOPs during inference can be expressed as $\sum_{i=1}^{p} f_i z_i$. This leads to the following ILP problem:

$$\max_{z \in \{0,1\}^p} \quad \sum_{i=1}^{p} I_i z_i, \ \text{ s.t. } \ \sum_{i=1}^{p} f_i z_i \leq F, \ \ \sum_{i=1}^{p} z_i \leq S. \ \text{(ILP)}$$

When the FLOP budget F is very large, the FLOP constraint in (ILP) becomes redundant, i.e., including the FLOP constraint does not affect the pruning decision. In this case, the problem reduces to magnitude pruning with only a sparsity constraint. On the other hand, if the sparsity constraint is redundant, (ILP) simplifies to magnitude pruning focused exclusively on FLOP cost. The (ILP) model allows for fine-grained control over FLOP and NNZ when both constraints are present. This scenario achieves the highest accuracy (under a fixed FLOP budget), as demonstrated in Section 5.1.2.

3.2 Solving the Relaxed Problem

Despite its seemingly simple structure, solving (ILP) is challenging due to its discrete nature. When both FLOP and sparsity constraints are considered, pruning becomes more challenging compared to Problem (1), as there is no closed-form optimal solution. While integer programming can, in principle, be solved for small-to-moderate size problems using powerful commercial solvers (e.g., Gurobi), we do not pursue this approach since the number of parameters p in the network can reach millions in practice, posing significant computational challenges.

For computational reasons, we consider a convex relaxation of (ILP), resulting in the following Linear Program (LP):

$$\max_{z \in [0,1]^p} \quad \sum_{i=1}^{p} I_i z_i, \ \text{ s.t. } \ \sum_{i=1}^{p} f_i z_i \leq F, \ \ \sum_{i=1}^{p} z_i \leq S. \ \text{(RLP)}$$

Problem (RLP) relaxes binary variables $\{z_i\}_{i=1}^p$ in (ILP) to continuous variables in $[0, 1]$. We will discuss in Theorem 3.2 how to retrieve a feasible binary solution from the relaxed problem and analyze the gap between that and the optimal solution to (ILP).

However, due to its large scale, solving (RLP) using commercial or open-source LP solvers is still time-consuming. To address this issue, we design an efficient custom solver for (RLP) by leveraging its dual problem, given by:

$$\min_{\lambda_1 \geq 0, \lambda_2 \geq 0} \ D(\lambda_1, \lambda_2) := S\lambda_1 + F\lambda_2 + \sum_{i=1}^{p} \max\{I_i - \lambda_1 - f_i\lambda_2, 0\}. \qquad (2)$$

A key observation is that for a fixed λ_2, minimization over λ_1 can be computed exactly as

$$\arg\min_{\lambda_1 \geq 0} D(\lambda_1, \lambda_2) = \max\{(I - \lambda_2 f)_{(S)}, 0\}, \qquad (3)$$

where $I = (I_1, \ldots, I_p)$, and $(v)_{(S)}$ denote the S-th largest element of a vector v. Importantly, the resulting function $g(\lambda_2) := \min_{\lambda_1 \geq 0} D(\lambda_1, \lambda_2)$ is a one-dimensional convex function (refer to (Boyd and Vandenberghe, 2004, Chapter 3) for a proof). As a consequence, the dual problem (2) can be efficiently solved using a golden-section search method (Yao et al., 2007, Chapter 10). The details of this procedure are outlined in Algorithm 1.

Optimized sorting method The main computational load in each iteration of Algorithm 1 stems from two steps: identifying the S-th largest element in the vector $(I - \lambda_2 f)$ and calculating $g(\lambda_2)$ according to (3). Using conventional methods (e.g., quicksort) for calculating the S-th largest element demands $O(p)$ time. Leveraging Fact 2.1, we can partition $(I - \lambda_2 f)$ into L sorted arrays for any λ_2, with no added cost except initial preprocessing. We introduce a novel algorithm that leverages this structure to reduce

the time complexity of finding $(I - \lambda_2 f)_{(S)}$ and $g(\lambda_2)$ to $O\big(L(\log p)^2\big)$. In mny practical scenarios, L often remains under 100—in such cases we observe useful speedups by a factor of dozens through our novel approach. Readers can refer to Appendix A.1 for a detailed description of our proposed approach.

Algorithm 1 Solving Problem (2) via golden-section search

Require: NNZ budget S, FLOP budget F, magnitude I_i and FLOP cost f_i of each parameter, and accuracy ε.

Initialize $\lambda_2^{min} = 0$, $\lambda_2^{max} = \max_{i\in[p]}\{I_i/f_i\}$ and $\alpha = \frac{3-\sqrt{5}}{2}$.
Set $\lambda_2 = \lambda_2^{min} + \alpha(\lambda_2^{max} - \lambda_2^{min})$, $\lambda_2' = \lambda_2^{max} - \alpha(\lambda_2^{max} - \lambda_2^{min})$.
while $\lambda_2^{max} - \lambda_2^{min} > \varepsilon$ **do**
  Compute $g(\lambda_2)$ and $g(\lambda_2')$ via efficient method discussed in Appendix A.1.
  if $g(\lambda_2) \le g(\lambda_2')$ **then**
    Update $\lambda_2^{max} = \lambda_2'$, $\lambda_2' = \lambda_2$, and $\lambda_2 = \lambda_2^{min} + \alpha(\lambda_2^{max} - \lambda_2^{min})$.
  else
    Update $\lambda_2^{min} = \lambda_2$, $\lambda_2 = \lambda_2'$, and $\lambda_2' = \lambda_2^{max} - \alpha(\lambda_2^{max} - \lambda_2^{min})$.
  end if
end while

3.3 Theoretical Results

The following theorem presents the cost of solving Problem (2) by Algorithm 1.

Theorem 3.1. *Making use of Fact 2.1, each iteration of Algorithm 1 takes $O\big(L(\log p)^2\big)$ time complexity. Moreover, for $\epsilon > 0$, it takes $O\big(p\log p + L(\log p)^2\log(1/\varepsilon)\big)$ time to compute a ε-accurate solution of the dual problem (2).*

Since Algorithm 1 achieves a linear convergence rate, we can reasonably assume that obtaining the optimal solution for the dual problem requires minimal cost in practice. Theorem 3.2 ensures that, given an optimal solution to the dual problem, we can recover a high-quality feasible solution for the original integer program (ILP). The proofs for Theorem 3.1 and Theorem 3.2 can be found in Appendix A.

Theorem 3.2. *We assume Fact 2.1 holds true, and denote $L_f = \sum_{j=1}^{L} f^j$. Given an optimal solution of the dual problem (2), we can compute a feasible binary solution $\hat{z}$ to (ILP) with the following optimality gap:*

$$[Q_I^* - Q_I(\hat{z})]/Q_I^* \le \max\{L/S, L_f/F\}, \tag{4}$$

where Q_I^* is the optimal objective of (ILP).

In most network architectures, the value of S is often in the millions, while the number of layers (which serves as the upper bound of L) is limited to a few hundred. Hence, L/S is approximately 10^{-4}. Similarly, the value of L_f/F is typically of the order of 10^{-4}. This suggests that the relaxed problem approximates Problem (ILP) with a medium-to-high accuracy.

We note that the algorithm and theorems presented in this section apply to any choice of $\{I_i\}_{i=1}^p$ in (ILP) as long as $I_i \ge 0$. Hence, our framework can be potentially applied to impact-based pruning with FLOP and NNZ constraints by defining I_i as the impact of the i-th parameter.

4 OPTIMIZATION FORMULATION FOR PRUNING

In this section, we present our algorithmic framework `FALCON` for pruning a neural network given constraints on the model's sparsity (NNZ) and FLOPs. In Section 4.1, we formulate the pruning problem as a sparse regression problem with both ℓ_0 and weighted ℓ_0 constraints. In Section 4.2, we propose a discrete first-order method (DFO) to handle both sparsity and FLOP constraints. This DFO method requires repeatedly solving Problem (ILP) proposed in the last section.

4.1 Optimization Formulation with NNZ and FLOP Constraints

Our optimization-based framework builds upon earlier work (LeCun et al., 1989; Hassibi and Stork, 1992; Singh and Alistarh, 2020) that use a local model $\mathcal{L}$ around the pre-trained weights $\bar{w}$:

$$\mathcal{L}(w) = \mathcal{L}(\bar{w}) + \nabla\mathcal{L}(\bar{w})^\top(w-\bar{w}) + \frac{1}{2}(w-\bar{w})^\top\nabla^2\mathcal{L}(\bar{w})(w-\bar{w}) + O(\|w-\bar{w}\|^3). \tag{5}$$

By selecting appropriate gradient and Hessian approximations $g \approx \nabla\mathcal{L}(\bar{w}), H \approx \nabla^2\mathcal{L}(\bar{w})$ and disregarding higher-order terms, we derive $Q_{L_0}(w)$ as a local approximation of the loss $\mathcal{L}$:

$$Q_{L_0}(w) := \mathcal{L}(\bar{w}) + g^\top(w-\bar{w}) + \frac{1}{2}(w-\bar{w})^\top H(w-\bar{w}). \tag{6}$$

Following recent works (Singh and Alistarh, 2020; Benbaki et al., 2023), we approximate the gradient using the stochastic gradient on n samples:

$$g = (1/n)\sum_{i=1}^{n}\nabla\ell_i(\bar{w}) = (1/n)X^\top e \in \mathbb{R}^p. \tag{7}$$

We approximate the Hessian matrix using the empirical Fisher information derived from the same n samples, i.e. we take:

$$H = \frac{1}{n}\sum_{i=1}^{n}\nabla\ell_i(\bar{w})\nabla\ell_i(\bar{w})^\top = \frac{1}{n}X^\top X \in \mathbb{R}^{p\times p}, \tag{8}$$

where $X = [\nabla \ell_1(\bar{w}), \ldots, \nabla \ell_n(\bar{w})]^\top \in \mathbb{R}^{n \times p}$. It's important to observe that matrix H has a rank of n. In practice, p could reach millions while n remains below thousands. Leveraging the low-rank property of H offers significant reductions in memory usage and runtime, as elaborated in the following discussion.

Putting together the pieces, our optimization model is formulated as follows:

$$\begin{aligned} \min_{w} \quad & Q_L(w) := g^\top (w - \bar{w}) + \frac{1}{2}(w - \bar{w})^\top H (w - \bar{w}) \\ & \qquad + \frac{n\lambda}{2}\|w - \bar{w}\|^2, \\ \text{s.t.} \quad & \|w\|_0 \le S, \quad \|w\|_{0,f} \le F, \end{aligned} \tag{9}$$

Here, $\lambda \ge 0$ represents the strength of the ridge regularization, while $\|w\|_0 \le S$ and $\|w\|_{0,f} \le F$ are two combinatorial constraints related to NNZ and FLOP budgets, respectively.

By leveraging the low-rank structure of the Hessian matrix, we can express our proposed formulation (9) in a Hessian-free form. This eliminates the need to store the expensive full dense $p \times p$ Hessian matrix—instead, we only need to store a much smaller $n \times p$ matrix X. This results in a substantial reduction in memory usage. In contrast, another FLOPs-aware pruning method (Singh and Alistarh, 2020) employs a dense $p \times p$ matrix as an approximation of the Hessian, which can be prohibitively expensive in terms of both runtime and memory. For completeness, all details on the formulation are provided in Appendix B.1.

Problem (9) can be formulated as a mixed integer quadratic program which is challenging to solve. In contrast to previous studies that consider a single cardinality constraint, here we are dealing with an additional FLOP constraint. This necessitates developing new algorithms, as we discuss below.

4.2 A Modified DFO Method for Solving (9)

We outline the primary concepts behind our suggested algorithm for problem (9) and provide further details in Appendix B.2.

Our optimization approach is based on the Discrete First-order (DFO) method (Bertsimas et al., 2016), which optimizes (9) using an iterative process. Every iteration, referred to as a *DFO update*, concurrently updates the support and weights. We discuss below how this DFO update is equivalent to solving the integer program (ILP).

Taking advantage of the low-rank structure, we can bypass the need to compute the entire Hessian matrix, resulting in important cost savings. While the basic version of the DFO algorithm may be slow for problems involving millions of parameters, we enhance our algorithm's computational performance by incorporating an active set strategy and schemes to update the weights on nonzero weights after support stabilization. These enhancements substantially improve computational efficiency and solution quality, making our method suitable for large-scale network pruning problems.

DFO update The DFO algorithm operates by taking a gradient step with step-size τ^s from the current iteration and projecting it onto the feasible set (see update (12) below). For any vector $\bar{x}$, the projection operator $P_{S,F}(\bar{x})$ is defined as

$$P_{S,F}(\bar{x}) = \arg\min_{x} \ \|x - \bar{x}\|^2 \ \text{ s.t. } \ \|x\|_0 \le S, \ \ \|x\|_{0,f} \le F. \tag{10}$$

$P_{S,F}(\bar{x})$ computes the feasible point to problem (9) that is closest to $\bar{x}$. Crucially, the projection can be efficiently solved using the ILP discussed in Section 3, as stated in the following lemma:

Lemma 4.1. *Solving the projection problem* (10) *is equivalent to solving* (ILP) *with* $I_i = (\bar{x}_i)^2$ *for* $i \in [p]$. *Given a solution* z^* *of* (ILP),

$$x_i = \bar{x}_i z_i^*, \ \forall i \in [p] \tag{11}$$

provides a solution to the projection problem (10).

The proof of the Lemma is provided in Appendix A.3. By our efficient approach (i.e., Algorithm 1) for (ILP), we can solve the projection problem to near-optimality quickly.

Applying DFO to problem (9) results in the following update:

$$w^{t+1} = \mathsf{DFO}(w^t, \tau^s) := P_{S,F}\left(w^t - \tau^s \nabla Q_L(w^t)\right) \tag{12}$$

where $\tau^s > 0$ is a suitable stepsize. Due to the low-rank structure of H, matrix-vector multiplications with H in $\nabla Q_L(w^t)$ can be performed by operating on matrix $X \in \mathbb{R}^{n \times p}$ with cost $O(np)$—see Appendix B.2 for details.

Convergence result Our DFO method can be viewed as an inexact proximal gradient method. At each iteration, we convert the projection operation in (12) into (ILP) (this is an exact reformulation) and solve (ILP) approximately via Algorithm 1. As per Gu et al. (2018), for convergence, we need the error in the approximate solution to (ILP) to be summable (across the iterations t). This can be achieved for example, by solving (ILP) to a higher accuracy (as t increases). The latter can be done with off-the-shelf IP solvers with which we can control the gap from the optimal objective value. In practice, we would need an IP solver to refine (or improve) the rounding-based solution derived from Algorithm 1. In this paper we do not further investigate the (theoretical) convergence guarantees of our algorithm because our experimental results appear to suggest that applying (12) (with the projection operator solved

approximately by Algorithm 1) for a fixed number of iterations already yields solutions of high quality. For more detailed convergence properties of inexact proximal gradient methods, please refer to Schmidt et al. (2011); Gu et al. (2018).

Enhancing DFO efficiency To improve the efficiency of the proposed DFO method, we adopt an active set strategy (Nocedal and Wright, 1999; Hazimeh and Mazumder, 2020) that limits DFO updates to a small subset of variables on active set—this brings down the cost of every iteration from $O(pn)$ to approximately $O(Sn)$. We further accelerate the convergence of the DFO method by using a back-solve procedure to refine the nonzero coefficients. Detailed discussions on active set strategy and back-solve procedure are available in Appendix B.2.1.

A multi-stage procedure The DFO method introduced above delivers high-quality feasible solutions to optimization problem (9). We refer to this as a single-stage approach, as it optimizes the local quadratic approximation to the loss function $\mathcal{L}$ for *one* time. However, the performance of the resulting pruned network is sensitive to the quality of the local quadratic approximation, and we observe a degrading accuracy when we aim for high sparsity and low FLOPs. To mitigate this, we propose a multi-stage process called FALCON++ that operates on a better approximation of the loss $\mathcal{L}$. Here we iteratively refine the local quadratic models and solve them with gradually decreasing NNZ and FLOP budgets, thereby preserving efficiency while improving accuracy. The multistage approach extends the framework of Singh and Alistarh (2020) to incorporate both FLOP and NNZ constraints. Different from gradual pruning (Han et al., 2015), FALCON++ avoids the need for computationally intensive fine-tuning via SGD. We present the details of FALCON++ in Algorithm 2. Note that FALCON (i.e., the single stage approach) can be viewed as a special case of FALCON++ by setting the number of stages $T_0 = 1$ in Algorithm 2. More details of our multi-stage approach are provided in Appendix B.2.3.

Algorithm 2 FALCON++: a multi-stage procedure for pruning networks under both NNZ and FLOPs budgets

Require: The pre-trained weights $\bar{w}$, target NNZ budget S and FLOPs budget F, and the number of stages T_0.

1: Set $w^0 = \bar{w}$; construct sequences of parameters with decreasing NNZ and FLOPs budgets as follows:

$$S_1 \geq S_2 \geq \cdots \geq S_{T_0} = S; \;\; F_1 \geq F_2 \geq \cdots \geq F_{T_0} = F$$

2: **for** $t = 1, 2, \ldots, T_0$ **do**
3: At current solution w^{t-1}, calculate the stochastic gradient on a batch of n training points
4: Construct the objective $Q_L(w)$ in (9) using Hessian approximation (8) and gradient approximation (7).
5: Obtain a solution w^t to problem (9) with sparsity budget S_t and FLOPs budget F_t by performing DFO updates (12) with the active set strategy and back-solve procedure (see Appendix B.2 for details).
6: **end for**

5 NUMERICAL EXPERIMENTS

In this section, we compare our proposed algorithms with existing start-of-the-art approaches in two scenarios: (i) one-shot pruning in which the model is pruned only once after it has been fully trained, and (ii) gradual pruning, also known as re-training, which iteratively prunes and fine-tunes the model over a period of time. We evaluate our proposed framework FALCON on various pre-trained 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). Detailed information on the experimental setup and reproducibility can be found in Appendix C.1. Ablation studies and more experimental results are shown in Appendix C.2.

5.1 One-shot Pruning

5.1.1 Accuracy given the FLOP budget

To assess the effectiveness of our proposed frameworks FALCON and FALCON++, we compare them with several leading one-shot pruning methods. These include MP (Mozer and Smolensky, 1989), WF (with FLOP-aware pruning statistic) (Singh and Alistarh, 2020, Appendix S6), CHITA (Benbaki et al., 2023). All these approaches have been used in sparse pruning, though none of these methods consider directly minimizing both FLOPs and NNZs. For a fair comparison, the NNZ budget S for these methods was calibrated such that the pruned networks meet the FLOP budget.

Table 1 compares the test accuracy for pruned ResNet20, MobileNetV1, and ResNet50 under varying FLOP budgets. Our FALCON framework delivers considerably superior accuracy relative to existing methods. Furthermore, our multi-stage method: FALCON++, surpasses other techniques substantially, without incurring the extra cost of re-training.

5.1.2 Quantifying the usefulness of having both FLOP and NNZ budget constraints

Our framework FALCON, can handle both sparsity and FLOP constraints, thereby enabling practitioners to effectively set and achieve their model compression objectives while retaining model accuracy as much as possible. In principle, as we take into account both FLOPs and sparsity (NNZ) constraints, we expect FALCON to result in a model outperforming those pruned solely under FLOPs or sparsity

Table 1: The pruning performance (accuracy) of various methods on ResNet20, MobileNetV1, and ResNet50. The bracketed number in the FLOPs column indicates the proportion of FLOPs needed for inference in the pruned network versus the dense network. We take five runs for our single-stage (FALCON) and multi-stage (FALCON++) approaches and report the mean and standard error (in the brackets). The **first** and **second** best accuracy values are highlighted in bold.

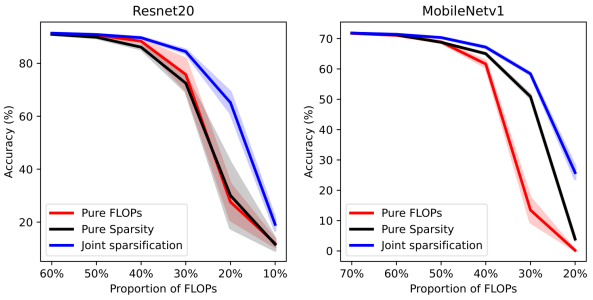
Network	FLOPs	MP	WF	CHITA	FALCON	FALCON++
ResNet20 on CIFAR10 (91.36%)	24.3M (60%)	88.89	91.13	90.95	**91.38(±0.10)**	**91.39(±0.10)**
	20.3M (50%)	85.95	90.42	89.86	**90.87(±0.09)**	**91.07(±0.13)**
	16.2M (40%)	80.42	87.83	86.10	**89.67(±0.18)**	**90.58(±0.17)**
	12.2M (30%)	58.78	77.43	72.48	**84.42(±0.70)**	**89.64(±0.26)**
	8.1M (20%)	15.04	39.21	30.07	**65.17(±3.95)**	**87.59(±0.16)**
	4.1M (10%)	10.27	12.31	11.61	**19.14(±2.25)**	**81.60(±0.37)**
MobileNetV1 on ImageNet (71.95%)	398M (70%)	70.89	71.70	71.82	**71.83(±0.05)**	**71.91(±0.01)**
	341M (60%)	67.34	70.95	71.25	**71.42(±0.04)**	**71.50(±0.04)**
	284M (50%)	51.13	68.32	68.78	**70.35(±0.10)**	**70.66(±0.06)**
	227M (40%)	14.85	55.76	65.03	**67.18(±0.24)**	**68.89(±0.03)**
	170M (30%)	0.65	18.72	50.89	**58.40(±0.31)**	**64.74(±0.13)**
	113M (20%)	0.10	0.19	3.92	**25.82(±2.09)**	**53.84(±0.11)**
ResNet50 on ImageNet (77.01%)	2.3G (60%)	75.38	76.67	76.56	**76.86(±0.02)**	**76.89(±0.03)**
	2.0G (50%)	70.85	75.76	75.22	**76.39(±0.02)**	**76.46(±0.07)**
	1.6G (40%)	62.37	72.65	73.32	**75.28(±0.05)**	**75.64(±0.06)**
	1.2G (30%)	22.43	58.78	64.05	**71.54(±0.15)**	**73.49(±0.15)**
	817M (20%)	0.56	5.39	12.88	**54.27(±0.44)**	**67.08(±0.20)**
	408M (10%)	0.10	0.10	0.10	**0.46(±0.07)**	**47.63(±0.38)**

constraints.

To illustrate this, we evaluate FALCON under a fixed FLOP budget (F_0) across three distinct scenarios. (i) Pure FLOP constraint: we set the FLOP budget to $F = F_0$ and the NNZ budget to $S = \infty$; (ii) Pure sparsity constraint: we fix $F = \infty$ and find the NNZ budget S so that the FLOPs of the resulting network precisely equals F_0; (iii) Joint sparsification: we choose $F = F_0$ and S optimally to maximize accuracy. As depicted in Figure 2, joint sparsification notably improves accuracy under a fixed FLOP budget compared to models pruned only based on a FLOP or NNZ constraint.

It appears to us that the effectiveness of our approach arises from its ability to attain relatively uniform sparsity across the layers of the pruned network. In Figure 3, we illustrate the sparsity of each group[2] in the Resnet50 model, pruned under a fixed FLOP budget (20% of total FLOPs) using (i) pure FLOP constraint, (ii) pure sparsity constraint, and (iii) joint sparsification. The joint sparsification approach strikes a nice balance between FLOP and NNZ constraints, leading to more even sparsity distribution across groups. This balanced sparsity distribution enhances accuracy, as demonstrated by Kusupati et al. (2020); Singh and Alistarh (2020).

[2]As per Fact 2.1, parameters within the same layer have the same FLOPs cost and are categorized within the same group. Hence, we use group sparsity for a clearer comparison.



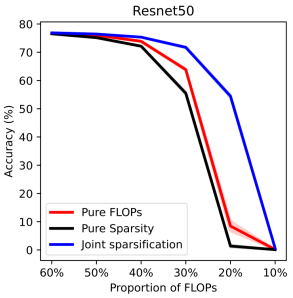
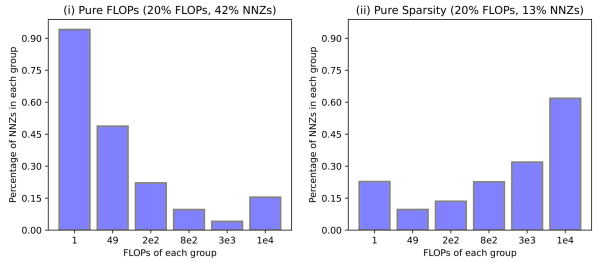


Figure 2: Accuracy of the network pruned by FALCON across three scenarios: pure FLOP constraint, pure sparsity constraint, and joint sparsification. The error bar represents the standard error over five runs.



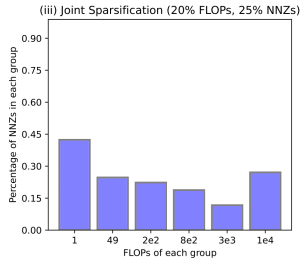


Figure 3: Sparsity of each group of Resnet50 pruned under a fixed FLOP budget (20% of total FLOPs) by FALCON across three scenarios: pure FLOP constraint, pure sparsity constraint, and joint sparsification.

5.2 Gradual Pruning

In addition to evaluating our algorithms within the context of one-shot pruning methods, we have also integrated them into the gradual pruning framework (Gale et al., 2019). This framework employs a gradual, multi-step pruning process interspersed with Stochastic Gradient Descent (SGD) training epochs. We compare our approach against WF (with FLOP-aware pruning statistic) (Singh and Alistarh, 2020, Appendix S6), GMP (Gale et al., 2019), STR (Kusupati et al., 2020), RIGL (Evci et al., 2020), DNW (Wortsman et al., 2019), and SNFS (Dettmers and Zettlemoyer, 2020). To match the FLOPs and NNZ of other baselines, we run FALCON with various NNZ and FLOP budgets, yielding

different accuracies for each budget.

Table 2: Results of gradually pruning MobilenetV1 (top) and ResNet50 (bottom), comparing FALCON to other baselines. FALCON numbers are averaged over two runs. WF numbers are in the appendix of Singh and Alistarh (2020); numbers for other baselines are taken from Kusupati et al. (2020).

Method	Sparsity (%)	FLOPs	Top-1 Acc (%)	NNZ
MobilenetV1	0 (dense)	569M	71.95	4.21M
GMP	74.11	163M	67.70	1.09M
STR	75.28	101M	68.35	1.04M
WF	75.28	101M	69.26	1.04M
FALCON	**75.28**	**101M**	**69.50**	**1.04M**
WF	75.28	92M	68.69	1.04M
FALCON	**75.28**	**92M**	**69.22**	**1.04M**
GMP	89.03	82M	61.80	0.46M
STR	85.80	55M	64.83	0.60M
FALCON	**90.00**	**55M**	**65.86**	**0.44M**
STR	90.00	40M	61.51	0.44M
FALCON	**92.97**	**40M**	**61.75**	**0.30M**

Method	Sparsity (%)	FLOPs	Top-1 Acc (%)	NNZ
ResNet50	0 (dense)	4.09G	77.01	25.6M
GMP	90.00	409M	73.91	2.56M
DNW	90.00	409M	74.00	2.56M
SNFS	90.00	1.63G	72.90	2.56M
SNFS + ERK	90.00	960M	72.90	2.56M
RigL	90.00	515M	72.00	2.56M
RigL + ERK	90.00	960M	73.00	2.56M
STR	90.55	341M	74.01	2.41M
WF	90.23	335M	74.34	2.49M
FALCON	**90.55**	**335M**	**74.72**	**2.41M**
GMP	95.00	204M	70.59	1.28M
DNW	95.00	204M	68.30	1.28M
RigL*	95.00	317M	67.50	1.28M
RigL + ERK	95.00	∼600M	70.00	1.28M
STR	95.03	159M	70.40	1.27M
FALCON	**95.03**	**159M**	**71.81**	**1.27M**
GMP	98.00	82M	57.90	0.51M
DNW	98.00	82M	58.20	0.51M
STR	98.22	68M	59.76	0.45M
FALCON	**98.22**	**68M**	**63.78**	**0.45M**

We carry out the gradual pruning evaluation on MobileNetV1 (4.2M parameters) and ResNet50 (25.6M parameters), and present the results in Table 2. We report the accuracy numbers of the competing methods from Singh and Alistarh (2020) and Kusupati et al. (2020)—they report numbers with varying FLOP and sparsity configurations. To facilitate a fair comparison, we tune FALCON so that it has the *lowest* FLOPs and NNZs compared to the other methods — we then observe that FALCON still achieves superior accuracy compared to the others.

We note that in gradual pruning, due to retraining, the fine-tuned model's accuracy closely mirrors that of the dense model. Hence, we do not expect a large absolute change in accuracy—this phenomenon is also observed in recent studies (Benbaki et al., 2023; Singh and Alistarh, 2020). However, in terms of relative accuracy drop, FALCON shows a significant improvement. Specifically, on a relative scale, the drop in accuracy of FALCON (fine-tuned model versus the dense model) is $14\% \sim 26\%$ smaller than that of WF and STR (our closest competitors) across different settings for Resnet50.

6 Conclusions and Discussion

In this work, we present an efficient optimization framework FALCON for network pruning that considers both FLOP and sparsity constraints. Our algorithm builds on our custom efficient solver for approximately solving a structured integer linear program (ILP). By leveraging the low-rank structure of the optimization problem and employing advanced combinatorial optimization strategies, we significantly improve our algorithm in terms of both runtime and memory. Our experiments confirm that FALCON outperforms existing pruning methods in performance under the same FLOP budgets, and its integration into gradual pruning frameworks leads to highly accurate networks with reduced FLOPs and NNZs.

FALCON is capable of pruning networks to specified FLOP and NNZ budgets while maintaining accuracy (as much as possible), achieving a good balance between inference time and memory usage. This offers the practitioner to set precise compression targets to maximize resource usage for high-performing models. In the future, we seek to expand the FALCON framework to incorporate more flexible resource constraints, potentially enhancing its flexibility and effectiveness across various computing scenarios.

Acknowledgments

This research is supported in part by grants from the Office of Naval Research (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. We thank Shibal Ibrahim for his helpful discussions.

References

Riade Benbaki, Wenyu Chen, Xiang Meng, Hussein Hazimeh, Natalia Ponomareva, Zhe Zhao, and Rahul Mazumder. Fast as CHITA: Neural network pruning with combinatorial optimization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 2031–2049. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/benbaki23a.html.

Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The annals of statistics*, 44(2):813–852, 2016.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511804441.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance, 2020. URL https://openreview.net/forum?id=ByeSYa4KPS.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

Bin Gu, De Wang, Zhouyuan Huo, and Heng Huang. Inexact proximal gradient methods for non-convex and non-smooth optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1539–1547, 2020.

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

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44 (3):243–254, 2016.

Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. *Advances in neural information processing systems*, 1, 1988.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Kenneth Lange. *MM optimization algorithms*. SIAM, 2016.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

A Woodbury Max. Inverting modified matrices. In *Memorandum Rept. 42, Statistical Research Group*, page 4. Princeton Univ., 1950.

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

Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

Mark Schmidt, Nicolas Roux, and Francis Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *Advances in neural information processing systems*, 24, 2011.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33:18098–18109, 2020.

Raphael Tang, Ashutosh Adhikari, and Jimmy Lin. Flops as a direct optimization objective for learning sparse neural networks. *arXiv preprint arXiv:1811.03060*, 2018.

Tom Veniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3492–3500, 2018.

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. *Discovering Neural Wirings*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Yu-Cheng Wu, Chih-Ting Liu, Bo-Ying Chen, and Shao-Yi Chien. Constraint-aware importance estimation for global filter pruning under multiple resource constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 686–687, 2020.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5687–5695, 2017.

Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pages 25668–25683. PMLR, 2022.

Michael Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Sy1iIDkPM.

A Proofs of Main Theorems in Section 3 and Auxiliary Results

A.1 Proof of Theorem 3.1

Proof. Our proof centers around the time complexity for each iteration of Algorithm 1. In each iteration, the algorithm computes the value of

$$g(\lambda_2) = \min_{\lambda_1 \geq 0} D(\lambda_1, \lambda_2) = D\big(\max\{(I - \lambda_2 f)_{(S)}, 0\}, \lambda_2\big) \tag{13}$$

for a given $\lambda_2 \geq 0$. It is clear that the primary computational expense lies in determining $(I - \lambda_2 f)_{(S)}$, which represents the S-th largest element of $I - \lambda_2 f$. Utilizing conventional algorithms (e.g., quicksort) for calculating $(I - \lambda_2 f)_{(S)}$ requires $O(p)$ time. However, we will demonstrate that the time complexity can be reduced by leveraging the structure of vector f.

Notably, Fact 2.1 states that parameters within the same group have identical FLOP costs. This implies that the size relationship between $(I - \lambda_2 f)_i$ and $(I - \lambda_2 f)_{i'}$ remains constant irrespective of the value of λ_2, provided i and i' belong to the same group C_j (for some $j \in [L]$). Hence, the values in each group can be pre-sorted such that for any value of λ_2, the elements in $(I - \lambda_2 f)$ can be divided into L sorted arrays without additional computational cost.

We propose Algorithm 3, which capitalizes on this structure to find the S-th largest element of $(I - \lambda_2 f)$ efficiently. The core idea of Algorithm 3 is to utilize the structure of sorted arrays to partition $I - \lambda_2 f$ based on a pivot element efficiently. This allows for recursively minimizing the search space to the desired S-th largest element by removing the unnecessary portions of the array. In this algorithm, the pivot element a_{σ_t} is wisely chosen to ensure that the number of elements in the search space is reduced by a constant factor at each recursive step.

It is straightforward to verify that Algorithm 3 consistently maintains the desired S-th largest element in the search space, thereby assuring its correctness. Now, we provide a formal estimation of the time complexity of Algorithm 3. Note that sorting a vector of length p requires $O(p \log p)$ time, and partitioning a sorted array of length p based on a value takes $O(\log p)$ time. The time complexity for line 1-7 in Algorithm 3 is thus

$$O(L \log L) + O\left(\sum_{j=1}^{L} \log(|v_j|)\right) \leq O(L \log L) + O(L \log p) = O(L \log p), \tag{14}$$

Next, we present the following lemma:

Lemma A.1. *The arrays $\{u_j\}_{j=1}^L$ and $\{s_j\}_{j=1}^L$ generated in Algorithm 3 satisfies*

$$\sum_{j=1}^{L} |u_j| \geq p/4, \ \sum_{j=1}^{L} |s_j| \geq p/4. \tag{15}$$

Proof. From the fact that $a_{\sigma_1} \geq a_{\sigma_2} \geq \cdots \geq a_{\sigma_L}$ and the definition of t and $\{u_j\}_{j=1}^L$, we derive

$$\begin{aligned} \sum_{j=1}^{L} |u_j| &= \big|\big\{a \in \cup_{j=1}^{L} \{v_j\} \big| a \geq a_{\sigma_t}\big\}\big| \\ &\geq \sum_{j=1}^{t} \big|\big\{a \in \ v_{\sigma_j} \big| a \geq a_{\sigma_j}\big\}\big| \\ &\geq \frac{1}{2} \sum_{j=1}^{t} |v_{\sigma_j}| \geq p/4. \end{aligned} \tag{16}$$

Likewise, given $a_{\sigma_1} \geq a_{\sigma_2} \geq \cdots \geq a_{\sigma_L}$ and the definition of t and $\{s_j\}_{j=1}^L$, we can deduce that

$$\begin{aligned}\sum_{j=1}^{L} |s_j| &= \left|\left\{a \in \cup_{j=1}^{L}\{v_j\} \middle| a \leq a_{\sigma_t}\right\}\right| \\ &\geq \sum_{j=t}^{L} \left|\left\{a \in \ v_{\sigma_j} \middle| a \leq a_{\sigma_j}\right\}\right| \\ &\geq \frac{1}{2}\sum_{j=t}^{L} |v_{\sigma_j}| = \frac{p}{2} - \frac{1}{2}\sum_{j=1}^{t-1} |v_{\sigma_j}| \geq p/4.\end{aligned} \tag{17}$$

This concludes the proof. □

From Lemma A.1, it follows that the number of elements remaining in sorted arrays decreases by at least $1/4$ after each recursion. Therefore, Algorithm 3 is able to find the S-th largest element in at most $O(\log p)$ recursive steps. Hence, the overall time complexity for Algorithm 3 is $O\big(L(\log p)^2\big)$.

Once we have computed the $(I - \lambda_2 f)_{(S)}$, $D\big(\max\{(I - \lambda_2 f)_{(S)}, 0\}, \lambda_2\big)$ in (13) can be calculated in $O(L \log p)$ time if we pre-compute and store the cumulative sum of L sorted arrays. Therefore, $g(\lambda_2)$ can be evaluated in $O\big(L(\log p)^2\big)$ time for any given λ_2.

Finally, we analyze the convergence of Algorithm 1. The algorithm employs the Golden-section search method, which, as described in (Yao et al., 2007, Chapter 10), guarantees that λ_2^{max} and λ_2^{min} consistently provide a valid upper and lower bound for the optimal value $\lambda_2^* = \arg\min_{\lambda_2 \geq 0} g(\lambda_2)$. At each iteration, the difference between the bounds, $\lambda_2^{max} - \lambda_2^{min}$, is multiplied by a factor of $(\sqrt{5} - 1)/2$. Consequently, it takes $O(\log(1/\varepsilon))$ iterations to obtain an ε-accurate solution. To conclude, the overall time complexity for Algorithm 1 to get a ε-accurate solution is $O\big(p \log p + L(\log p)^2 \log(1/\varepsilon)\big)$, where the term $O(p \log p)$ comes from pre-processing.

Algorithm 3 Largesort$(\{v_j\}_{j=1}^L, S)$: Calculate the S-th largest element in L sorted arrays

Require: L sorted arrays $v_1, v_2, \ldots, v_L$ with length denoted as $|v_1|, |v_2|, \ldots, |v_L|$, and an integer $S \in \left[1, p := \sum_{j=1}^L |v_i|\right]$

Let a_j be the median of array $v_j, \forall j \in [L]$.
Sort $\{a_j\}_{j=1}^L$ to obtain $a_{\sigma_1} \geq a_{\sigma_2} \geq \cdots \geq a_{\sigma_L}$.
Let $t = \arg\min \left\{t \in [L] \middle| \sum_{j=1}^t |v_{\sigma_j}| \geq p/2\right\}$.
for $j = 1, 2, \ldots, L$ **do**
    Split v_j into two sort arrays u_j, u_j' such that $a \in v_j$ belongs to u_j if and only if $a \geq a_{\sigma_t}$.
    Split v_j into two sort arrays s_j, s_j' such that $a \in v_j$ belongs to s_j if and only if $a \leq a_{\sigma_t}$.
end for
if $\sum_{j=1}^L |u_j| \leq S - 1$ **then**
    Return Largesort$(\{u_j'\}_{j=1}^L, S - \sum_{j=1}^L |u_j|)$
else if $\sum_{j=1}^L |s_j| \geq p - S$ **then**
    Return Largesort$(\{s_j'\}_{j=1}^L, S)$
else
    Return a_{σ_t}
end if

□

A.2 Proof of Theorem 3.2

Proof. Given the optimal dual solution, $(\lambda_1^*, \lambda_2^*)$, we first discuss how to obtain an optimal primal solution for the relaxed linear programming problem expressed in (RLP).

Note that λ_1 and λ_2 are Lagrangian multipliers associated with the constraints $\sum_{i=1}^p z_i \leq S$ and $\sum_{i=1}^p f_i z_i \leq F$, respectively. If $\lambda_2^* = 0$, then the constraint $\sum_{i=1}^p f_i z_i \leq F$ is inactive, allowing us to disregard this constraint in the primal

problem and solve it by sorting I_i's to obtain the primal optimal solution z^*. If $\lambda_1^* = 0$, then the constraint $\sum_{i=1}^p z_i \le S$ is inactive. In this case, we may sort I_i/f_i to get the optimal primal solution z^*.

We now consider the case where both λ_1^* and λ_2^* are greater than 0. The KKT condition implies that the primal optimal solution z^* satisfies:

$$\begin{aligned} z_i^* \in & \begin{cases} \{1\}, & \text{if } i \in \mathcal{Z}_1 := \left\{ i \in [p] \middle| I_i - \lambda_1^* - f_i\lambda_2^* > 0 \right\}, \\ [0,1], & \text{if } i \in \mathcal{Z}_2 := \left\{ i \in [p] \middle| I_i - \lambda_1^* - f_i\lambda_2^* = 0 \right\}, \\ \{0\}, & \text{if } i \in \mathcal{Z}_3 := \left\{ i \in [p] \middle| I_i - \lambda_1^* - f_i\lambda_2^* < 0 \right\}, \end{cases} \\ 0 = & \lambda_1^*(S - \sum_{i=1}^p z_i^*), \\ 0 = & \lambda_2^*(F - \sum_{i=1}^p f_i z_i^*). \end{aligned} \tag{18}$$

From this, we can conclude that $\sum_{i=1}^p z_i^* = S$ and $\sum_{i=1}^p f_i z_i^* = F$ hold. It remains to determine the values of z_i^* for $i \in \mathcal{Z}_2$. We fix $z_i^* = 1$ for $i \in \mathcal{Z}_1$ and $z_i^* = 0$ for $\mathcal{Z}_3$ in (RLP), and solve the resulting problem

$$\begin{aligned} \max_z \quad & \sum_{i \in \mathcal{Z}_2} I_i z_i, \\ \text{s.t.} \quad & \sum_{i \in \mathcal{Z}_2} f_i z_i = F - \sum_{i \in \mathcal{Z}_1} f_i, \ \sum_{i \in \mathcal{Z}_2} z_i = S - |\mathcal{Z}_1|, \\ & z_i \in [0,1] \ \ \forall i \in [p]. \end{aligned} \tag{19}$$

Since $I_i - \lambda_1^* - f_i\lambda_2^* = 0$ for any $i \in \mathcal{Z}_2$, the above problem is equivalent to the following feasibility problem and can be readily solved.

$$\begin{aligned} \max_z \quad & \sum_{i \in \mathcal{Z}_2} I_i z_i = \lambda_2^*(F - \sum_{i \in \mathcal{Z}_1} f_i) + \lambda_1^*(S - |\mathcal{Z}_1|), \\ \text{s.t.} \quad & \sum_{i \in \mathcal{Z}_2} f_i z_i = F - \sum_{i \in \mathcal{Z}_1} f_i, \ \sum_{i \in \mathcal{Z}_2} z_i = S - |\mathcal{Z}_1|, \\ & z_i \in [0,1] \ \ \forall i \in [p]. \end{aligned} \tag{20}$$

Given an optimal primal solution z^* of the relaxed problem (RLP), we now focus on retrieving a feasible solution $\hat{z}$ for the MIP problem (ILP) and analyzing its quality. Denote $\mathcal{Z}_1^* := \{i | z_i^* = 1\}$, $\mathcal{Z}_2^* := \{i | z_i^* \in (0,1)\}$, and $\mathcal{Z}_3^* := \{i | z_i^* = 0\}$.The KKT condition (18) implies that $I_i = \lambda_1^* + f_i\lambda_2^*$ for $i \in \mathcal{Z}_2^*$.

Recall that in Assumption 2.1, all parameters are partitioned into L groups $C_1, \ldots, C_L$. Suppose for some $j \in [L]$ there exists $i, i' \in C_j \cap \mathcal{Z}_2^*$ with $i \neq i'$. From the KKT condition and $f_i = f_{i'}$, we have

$$I_i = \lambda_1^* + f_i\lambda_2^* = \lambda_1^* + f_{i'}\lambda_2^* = I_{i'}. \tag{21}$$

Therefore, we can replace the value of z_i^* and $z_{i'}^*$ with $\min\{1, z_i^* + z_{i'}^*\}$ and $\max\{0, z_i^* + z_{i'}^* - 1\}$, without altering the objective or violating the constraints. This implies that after adjustment, z^* is still an optimal primal solution. By adjusting z^*, we can reduce the size of $|C_j \cap \mathcal{Z}_2^*|$ as long as it is no less than 2. Therefore, without loss of generality, we assume that $|C_j \cap \mathcal{Z}_2^*| \le 1$ for any $j \in [L]$. Now we set

$$\hat{z}_i = \begin{cases} 1 & \text{if } i \in \mathcal{Z}_1^*, \\ 0 & \text{if } i \in \mathcal{Z}_2^* \cup \mathcal{Z}_3^*. \end{cases} \tag{22}$$

Since $\sum_{i=1}^p \hat{z}_i \le \sum_{i=1}^p z_i^* \le S$ and $\sum_{i=1}^p f_i\hat{z}_i \le \sum_{i=1}^p f_i z_i^* \le F$, $\hat{z}$ is a feasible solution to the MIP problem (ILP). On the other hand, since z^* is the optimal solution of the relaxed problem of (ILP), $Q_I(z^*) = \sum_{i=1}^p I_i z_i$ provides an upper

bound of Q_I^*, the optimal objective of the MIP problem. Together with $D(\lambda_1^*, \lambda_2^*) = \sum_{i=1}^p I_i z_i^* \geq S\lambda_1^* + F\lambda_2^*$, we conclude that

$$\begin{aligned}
\frac{Q_I^* - Q_I(\hat{z})}{Q_I^*} &\leq \frac{\sum_{i=1}^p I_i(z_i^* - \hat{z}_i)}{\sum_{i=1}^p I_i z_i^*}, \\
&\leq \frac{\sum_{i\in\mathcal{Z}_2^*} I_i}{D(\lambda_1^*, \lambda_2^*)}, \\
&= \frac{\sum_{i\in\mathcal{Z}_2^*} \lambda_1^* + f_i\lambda_2^*}{D(\lambda_1^*, \lambda_2^*)}, \\
&\leq \frac{\sum_{j=1}^L \lambda_1 + \sum_{j=1}^L f^j\lambda_2^*}{D(\lambda_1^*, \lambda_2^*)}, \\
&\leq \frac{L\lambda_1^* + L_f\lambda_2^*}{S\lambda_1^* + F\lambda_2^*}, \\
&\leq \max\left\{\frac{L}{S}, \frac{L_f}{F}\right\},
\end{aligned} \tag{23}$$

which completes the proof. □

A.3 Auxiliary results

In this subsection, we present Proposition A.2, which establishes the equivalence between magnitude pruning and solving Problem (1), along with the proof of Lemma 4.1.

A.3.1 Equivalence between magnitude pruning and Problem (1)

Proposition A.2. *Using magnitude pruning to prune the weight vector $\bar{w}$ with NNZ budget S is equivalent to solving the following cardinality-constrained problem*

$$\max_{z\in\{0,1\}^p} \quad Q_I(z) := \sum_{i=1}^p I_i z_i, \quad \textit{s.t.} \quad \sum_{i=1}^p z_i \leq S. \tag{24}$$

with $I_i = (\bar{w}_i)^2$ and setting the pruned weights as $w_i = \bar{w}_i z_i$ after Problem (1) is solved for z_i's.

Proof. MP selects a set of parameters with the least absolute values to remove while adhering to NNZ budget. Let $\mathcal{I}$ denote the set of indices of the S largest elements in $(\bar{w})^2$. For simplicity, we assume that such a set is unique; other cases can be handled similarly. Applying MP to prune the weight vector $\bar{w}$ with NNZ budget S results in a vector w expressed as

$$w_i = \begin{cases} \bar{w}_i & \text{if } i \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases} \tag{25}$$

On the other hand, $\{z_i\}_{i=1}^p$ are binary variables and $I_i \geq 0$ in Problem (24). Given the NNZ budget $\sum_{i=1}^p z_i \leq S$, the optimal solution of Problem (24) is

$$z_i = \begin{cases} 1 & \text{if } i \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

Therefore, setting $w_i = \bar{w}_i z_i$ results in a pruned weight vector identical to that produced by MP. □

A.3.2 Proof of Lemma 4.1

Proof. By setting $z_i = \mathbf{1}_{x_i\neq 0}$, the projection problem (10) can be reformulated as

$$\begin{aligned}
\underset{x\in\mathbb{R}^p, z\in\{0,1\}^p}{\arg\min} \quad & \sum_{i=1}^p (x_i - \bar{x}_i)^2, \\
\text{s.t.} \quad & \sum_{i=1}^p z_i \leq S, \quad \sum_{i=1}^p f_i z_i \leq F.
\end{aligned} \tag{27}$$

Note that once $\{z_i\}_{i=1}^p$ are fixed, the optimal choices of $\{x_i\}_{i=1}^p$ that minimize the objective are given by $x_i = \bar{x}_i z_i,\ \forall i \in [p]$. Therefore, we can replace x_i with $\bar{x}_i z_i$ in (27), which leads to the following equivalent problem:

$$\begin{aligned} \underset{z\in\{0,1\}^p}{\arg\min} \quad & \sum_{i=1}^p (\bar{x}_i z_i - \bar{x}_i)^2 = \sum_{i=1}^p (\bar{x}_i)^2(2 - 2z_i), \\ \text{s.t.} \quad & \sum_{i=1}^p z_i \le S, \quad \sum_{i=1}^p f_i z_i \le F. \end{aligned} \tag{28}$$

This problem is further equivalent to (ILP) with $I_i = (\bar{x}_i)^2\ \forall i \in [p]$. Hence, solving (ILP) with $I_i = (\bar{x}_i)^2\ \forall i \in [p]$ and setting $x_i = \bar{x}_i z_i,\ \forall i \in [p]$ provides a solution to the projection problem (10). □

B Details on Optimization Formulation and Algorithm

B.1 Optimization formulation details

Our optimization-based framework use a local model $\mathcal{L}$ around the pre-trained weights $\bar{w}$:

$$\mathcal{L}(w) = \mathcal{L}(\bar{w}) + \nabla\mathcal{L}(\bar{w})^\top (w - \bar{w}) + \frac{1}{2}(w - \bar{w})^\top \nabla^2 \mathcal{L}(\bar{w})(w - \bar{w}) + O(\|w - \bar{w}\|^3). \tag{29}$$

By selecting appropriate gradient and Hessian approximations $g \approx \nabla\mathcal{L}(\bar{w}), H \approx \nabla^2\mathcal{L}(\bar{w})$ and disregarding higher-order terms, we derive $Q_{L_0}(w)$ as a local approximation of the loss $\mathcal{L}$:

$$Q_{L_0}(w) := \mathcal{L}(\bar{w}) + g^\top (w - \bar{w}) + \frac{1}{2}(w - \bar{w})^\top H (w - \bar{w}). \tag{30}$$

Choices of Hessian approximation H. Previous works (Hassibi and Stork, 1992; Singh and Alistarh, 2020; Benbaki et al., 2023) have approximated the Hessian matrix using the empirical Fisher information matrix derived from n samples. It has been found that using a few hundred to one thousand samples is sufficient for Hessian estimation, resulting in a low-rank representation:

$$\widehat{H} = \frac{1}{n}\sum_{i=1}^n \nabla\ell_i(\bar{w})\nabla\ell_i(\bar{w})^\top = \frac{1}{n} X^\top X \in \mathbb{R}^{p\times p}, \tag{31}$$

where $X = [\nabla\ell_1(\bar{w}), \dots, \nabla\ell_n(\bar{w})]^\top \in \mathbb{R}^{n\times p}$. This low-rank structure is leveraged in (Benbaki et al., 2023) to circumvent the need for storing the full dense Hessian and to accelerate convergence.

Choices of gradient approximation g. Traditional pruning methods for neural networks typically assume that the pre-trained weights $\bar{w}$ represent a local optimum of the loss function $\mathcal{L}$ and thus set the gradient g to zero. However, in practice, the gradient of a pre-trained neural network's loss function may not be zero due to early stopping or approximate optimization (Yao et al., 2007). Additionally, pruning the local quadratic model at a general reference point, where the gradient is not zero, may yield more desirable results. Consequently, we propose approximating the gradient using the stochastic gradient, which leverages the same samples as those used for estimating the Hessian:

$$g = \frac{1}{n}\sum_{i=1}^n \nabla\ell_i(\bar{w}) = \frac{1}{n} X^\top e \in \mathbb{R}^p, \tag{32}$$

where $e \in \mathbb{R}^n$ is a vector of all ones.

Objective construction Utilizing the Hessian approximation (8) and gradient approximation (32), we formulate the network pruning as an optimization problem that minimizes $Q_{L_0}(w)$ while adhering to NNZ and FLOPs constraints:

$$\min_w \ Q_{L_0}(w) \qquad \text{s.t.} \quad \|w\|_0 \le S, \quad \|w\|_{0,f} \le F. \tag{33}$$

Based on our empirical observations, the performance of the pruned model is heavily dependent on the accuracy of the quadratic approximation $Q_{L_0}(w)$ for the loss function. As this approximation is local, it is imperative to ensure that the

weights w during pruning remain in close proximity to the initial weights $\bar{w}$. Thus, we incorporate a ridge-like regularizer of the form $\|w-\bar{w}\|^2$ into the objective in (33), resulting in the following problem:

$$\begin{aligned} \min_{w} \quad & Q_L(w) := g^\top (w-\bar{w}) + \frac{1}{2}(w-\bar{w})^\top H (w-\bar{w}) + \frac{n\lambda}{2}\|w-\bar{w}\|^2, \\ \text{s.t.} \quad & \|w\|_0 \le S \quad \|w\|_{0,f} \le F, \end{aligned} \tag{34}$$

where $\lambda \ge 0$ is a parameter governing the strength of the regularization.

Special consideration for block approximation We employ a block-wise approximation $\widehat{H}_B$ of $\widehat{H}$, considering only limited-size blocks on the diagonal of $\widehat{H}$ and disregarding off-diagonal elements.We treat the set of variables corresponding to a single layer in the network as a block and uniformly subdivide these blocks such that the size of each block does not exceed a given parameter B_{size}. Based on our empirical observations, this block-wise approximation $\widehat{H}_B$ delivers a more accurate approximation of the Hessian matrix when compared to the original $\widehat{H}$.

Furthermore, we multiply the block-wise approximation $\widehat{H}_B$ by a constant factor ρ, ensuring that $\widehat{H} \preceq H := \rho \widehat{H}_B$. This guarantees that $Q_{L_0}(w)$ serves as an (approximate) upper bound of $L(w)$:

$$\begin{aligned} L(w) &\approx \mathcal{L}(\bar{w}) + g^\top (w-\bar{w}) + \frac{1}{2}(w-\bar{w})^\top \widehat{H} (w-\bar{w}) \\ &\le \mathcal{L}(\bar{w}) + g^\top (w-\bar{w}) + \frac{\rho}{2}(w-\bar{w})^\top \widehat{H}_B (w-\bar{w}) = Q_{L_0}(w). \end{aligned} \tag{35}$$

This procedure is inspired by the well-known majorization-minimization approach (Lange, 2016), where one minimizes an upper bound on the objective at each iteration. In our experiments, we observe that this approach can help mitigate the effects of imprecise Hessian and gradient approximations during the optimization procedure.

Given a disjoint partition $\{\mathcal{B}_j\}_{j=1}^K$ of $\{1,2,\dots,p\}$ and assuming blocks of size $|\mathcal{B}_1|\times|\mathcal{B}_1|,\dots,|\mathcal{B}_K|\times|\mathcal{B}_K|$ along the diagonal of $\widehat{H}_B$, we can represent H as:

$$H = \rho \widehat{H}_B = \rho\, \mathrm{Diag}(X_{\mathcal{B}_1}^\top X_{\mathcal{B}_1}, \dots, X_{\mathcal{B}_K}^\top X_{\mathcal{B}_K}), \tag{36}$$

where $X_{\mathcal{B}_j}$ is a submatrix of $X = [\nabla \ell_1(\bar{w}), \dots, \nabla \ell_n(\bar{w})]^\top$ with columns in $\mathcal{B}_j$. Consequently, our optimization formulation (34) is Hessian-free, necessitating only the storage of a matrix $X \in \mathbb{R}^{n\times p}$. Moreover, by leveraging the representation (36), the matrix-vector multiplications involving H in the DFO update

$$\begin{aligned} w^{t+1} = \mathsf{DFO}(w^t, \tau^s) &:= P_{S,F}\left(w^t - \tau^s \nabla Q_L(w^t)\right) \\ &= P_{S,F}\left(w^t - \tau^s\big(g + H(w^t-\bar{w}) + n\lambda(w^t-\bar{w})\big)\right), \end{aligned} \tag{37}$$

can be executed by operating on the matrix $X \in \mathbb{R}^{n\times p}$ with cost $O(np)$.

To conclude, our algorithm operates solely on the low-rank matrix X and circumvents the need to directly store and solve problem (34), which would be computationally demanding due to the considerable size of the $p \times p$ matrix H for large networks. Our approach results in significant improvements in both memory usage and runtime.

B.2 Algorithmic details

B.2.1 Active set updates

The active set strategy has demonstrated its effectiveness in various contexts by reducing complexity (Nocedal and Wright, 1999; Hazimeh and Mazumder, 2020). This section demonstrates how to apply an active set strategy to accelerate our DFO updates. We begin by projecting the pre-trained weight onto a set with NNZ budget $S' \ge S$ and FLOPs budget $F' \ge F$, obtaining $\hat{w} = P_{S',F'}(\bar{w})$. We then define the initial active set as $\mathcal{Z} = \mathrm{supp}(\hat{w}) := \{i | \hat{w}_i \neq 0\}$.

During each iteration, we limit DFO updates to the current active set $\mathcal{Z}$. Upon convergence, we execute a single DFO update on the entire vector to identify an improved solution w with $\mathrm{supp}(w) \not\subseteq \mathcal{Z}$. If no such w exists, the algorithm terminates; otherwise, we update $\mathcal{Z} \leftarrow \mathcal{Z} \cup \mathrm{supp}(w)$ and repeat the process. Algorithm 4 offers a detailed depiction of the active set method.

Algorithm 4 Active set with DFO: ACT-DFO$(w^0, \tau^s, T, \mathcal{Z}^0)$

Require: Initial solution w^0, stepsize τ^s, the number of DFO iterations T, and an initial set $\mathcal{Z}^0$.

for $t = 0, 1, \ldots$ **do**
  for $t' = 0, 1, \ldots, T$ **do**
    Perform DFO update $w^t|_{\mathcal{Z}^t} = \mathsf{DFO}(w^t|_{\mathcal{Z}^t}, \tau^s)$ restricted on $\mathcal{Z}^t$
  end for
  Find τ' via line search such that $w^{t+1} = \mathsf{DFO}(w^t, \tau')$ satisfies
    (i) $Q_L(w^{t+1}) < Q_L(w^t)$ (ii) $\mathrm{supp}(w^{t+1}) \not\subseteq \mathcal{Z}^t$
  if such τ' does not exist **then**
    break
  else
    $\mathcal{Z}^{t+1} \leftarrow \mathcal{Z}^t \cup \mathrm{supp}(w^{t+1})$
  end if
end for

B.2.2 Backsolve via Woodbury formula

As the problem's dimensionality increases, DFO updates becomes increasingly computationally demanding, even when employing an active set strategy. To address this challenge, we propose a backsolve approach that reduces complexity while maintaining a slightly suboptimal solution. The backsolve approach computes the optimal solution precisely on a restricted set. We initially apply DFO updates several times to obtain a feasible solution w and then restrict the problem to the set $\mathcal{Z} := \mathrm{supp}(w)$. Under this restriction, problem (34) simplifies to a quadratic problem without any constraint, and its minimizer is given by:

$$w^*_{\mathcal{Z}} = (n\lambda I + H_{\mathcal{Z}})^{-1}\big((H_{\mathcal{Z}} + n\lambda I)\bar{w}_{\mathcal{Z}} + g_{\mathcal{Z}}\big), \tag{38}$$

where $H_{\mathcal{Z}} \in \mathbb{R}^{|\mathcal{Z}|\times|\mathcal{Z}|}$ denotes a submatrix of H with rows and columns only in $\mathcal{Z}$.

Recall that $\{\mathcal{B}_j\}_{j=1}^K$ is a disjoint partition of $\{1, 2, \ldots, p\}$, and $H = \rho\widehat{H}_B$ is a block diagonal matrix with block sizes $|\mathcal{B}_1| \times |\mathcal{B}_1|, \ldots, |\mathcal{B}_K| \times |\mathcal{B}_K|$ along the diagonal, represented as:

$$H = \rho\widehat{H}_B = \rho\,\mathrm{Diag}(X_{\mathcal{B}_1}^\top X_{\mathcal{B}_1}, \ldots, X_{\mathcal{B}_K}^\top X_{\mathcal{B}_K}), \tag{39}$$

where $X_{\mathcal{B}_j}$ is a submatrix of $X = [\nabla\ell_1(\bar{w}), \ldots, \nabla\ell_n(\bar{w})]^\top$ with columns in $\mathcal{B}_j$. Based on this representation, $H_{\mathcal{Z}}$ can be expressed as

$$H_{\mathcal{Z}} = \rho\,\mathrm{Diag}(X_{\mathcal{B}'_1}^\top X_{\mathcal{B}'_1}, \ldots, X_{\mathcal{B}'_K}^\top X_{\mathcal{B}'_K}), \tag{40}$$

where $\mathcal{B}'_j = \mathcal{B}_j \cap \mathcal{Z}$, $\forall j \in [K]$.

By leveraging the low-rank representation (40) and utilizing Woodbury formula (Max, 1950), one can compute each sub-vector $w^*_{\mathcal{B}'_j}$ of $w^*_{\mathcal{Z}}$ in (38) using matrix-vector multiplications involving only $X_{\mathcal{B}'_j}$ (or its transpose) and one matrix-matrix multiplication via

$$\begin{aligned} w^*_{\mathcal{B}'_j} &= (n\lambda I + H_{\mathcal{B}'_j})^{-1}\big((H_{\mathcal{B}'_j} + n\lambda I)\bar{w}_{\mathcal{B}'_j} + g_{\mathcal{B}'_j}\big) \\ &= (n\lambda)^{-1}[I - \rho X_{\mathcal{B}'_j}^\top(n\lambda I + \rho X_{\mathcal{B}'_j} X_{\mathcal{B}'_j}^\top)^{-1} X_{\mathcal{B}'_j}] \cdot \big(n\lambda\bar{w}_{\mathcal{B}'_j} + \rho X_{\mathcal{B}'_j}^\top X_{\mathcal{B}'_j}\bar{w}_{\mathcal{B}'_j} + g_{\mathcal{B}'_j}\big). \end{aligned} \tag{41}$$

It takes $O(n^3 + n^2|\mathcal{B}'_j|)$ operations to compute $w^*_{\mathcal{B}'_j}$, thus the overall complexity for backsolve is $\sum_{j=1}^K O(n^3 + n^2|\mathcal{B}'_j|) = O(n^3K + n^2S)$. Our proposed backsolve procedure is detailed in Algorithm 5.

B.2.3 The multi-stage procedure

In this section, we present a multi-stage procedure that iteratively updates and solves local quadratic models, utilizing the BSO-DFO method introduced earlier as the inner solver.

Our multi-stage procedure employs a scheduler to gradually decrease the NNZ and FLOPs budget, taking small steps towards increased sparsity and reduced FLOPs at every stage. This cautious approach helps maintain the validity of the local quadratic approximation, ensuring that each step is based on accurate information.

Algorithm 5 Backsolve: BSO-DFO(w^0)

Require: Initial solution w^0.

1: Construct an initial active set $\mathcal{Z}^0$; determine stepsize τ^s and the number of DFO iterations T
2: Set $w = \texttt{ACT-DFO}(w^0, \tau^s, T, \mathcal{Z}^0)$
3: $\mathcal{Z} \leftarrow \text{supp}(w)$
4: **for** $j = 1, \ldots, K$ **do**
5: Set $\mathcal{B}'_j = \mathcal{B}_j \cap \mathcal{Z}$ and compute $w^*_{\mathcal{B}'_j}$ according to (41).
6: **end for**
7: Concatenating sub-vectors $\{w^*_{\mathcal{B}'_j}\}_{j=1}^K$ to form a complete vector w^*.

The multi-stage method leverages the efficiency of the BSO-DFO algorithm to achieve a balance between computational efficiency and model accuracy. By iteratively solving precise approximations of the true loss function, our approach can efficiently produce pruned networks with superior performance compared to single-stage techniques. The details of the proposed multi-stage procedure can be found in Algorithm 6.

Algorithm 6 A multi-stage procedure for pruning networks under both NNZ and FLOPs budgets

Require: Target NNZ budget S and FLOPs budget F, and the number of stages T_0.

1: Set $w^0 = \bar{w}$; construct sequences of parameters with decreasing NNZ and FLOPs budgets as follows:

$$S_1 \geq S_2 \geq \cdots \geq S_{T_0} = S; \;\; F_1 \geq F_2 \geq \cdots \geq F_{T_0} = F \tag{42}$$

2: **for** $t = 1, 2, \ldots, T_0$ **do**
3: At current solution w^{t-1}, calculate the gradient based on a batch of n data points
4: Construct the objective $Q_L(w)$ as in (34), using Hessian approximation (36) and gradient approximation (32).
5: Obtain a solution w^t to problem (33) with sparsity budget S_t and FLOPs budget F_t by invoking $\texttt{BSO-DFO}(w^{t-1})$.
6: **end for**

C Experimental details

C.1 Experimental setup

C.1.1 One-shot pruning experiments

All experiments for one-shot pruning were carried on a computing cluster. Experiments were run on an Intel Xeon Gold 6248 machine with 40 CPUs and one GPU.

Algorithmic setting and hyper-parameters for FALCON (single-stage)

We apply the BSO-DFO algorithm (Algorithm 5) to prune ResNet20, MobileNetV1, and ResNet50. The parameters of BSO-DFO are set as follows: $\tau^s = 10^{-3}$, $T = 1$ and $\mathcal{Z}_0 = \text{supp}(P_{2S,2F}(\bar{w}))$. For ResNet20 and MobileNetV1, we use $n = 1000$ samples for Hessian approximation (31) and gradient estimation (32); for ResNet50, we use $n = 500$ samples. Each block of the block-wise Hessian approximation has size $B_{size} = 2000$. We run our proposed methods with a scaling factor ρ of one and a ridge value λ ranging from 10^{-6} to 10^{-2} for each network and FLOPs budget. All results are averaged over 5 runs, and the highest accuracy results are shown.

Algorithmic setting and Hyper-parameters for FALCON++ (multi-stage)

We apply (Algorithm 6 with the BSO-DFO algorithm as an inner solver to prune ResNet20, MobileNetV1, and ResNet50. The number of stages T_0 is set to be 20. We run our proposed methods with a scaling factor ρ range from 10^2 to 10^4 and a ridge value λ ranging from $10^{-5}\rho$ to $10^{-2}\rho$ for each network and FLOPs budget. All results are averaged over 5 runs, and the highest accuracy results are shown. The remaining parameters are the same as those in our single-stage method FALCON.

C.1.2 Gradual pruning experiments

All experiments for one-shot pruning were carried on a computing cluster. Experiments for MobileNetV1 were run on an Intel Xeon Platinum 6248 machine with 48 CPUs and 4 GPUs; experiments for ResNet50 were run on four Intel Xeon Platinum 6248 machines with a total of 160 CPUs and 4 GPUs.

In our gradual pruning experiments, we performed 100 epochs of SGD training with intermittent pruning steps. We employed the polynomial schedule introduced by Zhu and Gupta (2018) as the pruning schedule to gradually reduce the NNZ and FLOPs budget in each pruning step until reaching our target budget. We utilize the `BSO-DFO` algorithm as the pruning method.

The learning rate for each epoch e between two pruning steps that occur at epochs e_1 and e_2 is defined as:

$$\textit{min\_lr} + 0.5 \times (\textit{max\_lr} - \textit{min\_lr}) \left(1 + \cos\left(\pi \frac{e - e_1}{e_2 - e_1}\right)\right) \tag{43}$$

We used a momentum of 0.9 and a weight decay penalty of 3.75×10^{-5} (taken from Kusupati et al. (2020)).

For MobileNetV1, we pruned the network 8 times during training, with 12 epochs between pruning steps. For the first 84 epochs, we set *min_lr* $= 10^{-5}$ and *max_lr* $= 0.1$. For remaining epochs, we set *min_lr* $= 10^{-5}$ and *max_lr* $= 0.05$. The batch size was set to 4×256.

For ResNet50, we pruned the network 7 times during training, with 12 epochs between pruning steps. The *min_lr* and *max_lr* values were kept constant as 10^{-5} and 0.1, respectively. The batch size was set to 8×256.

C.2 Ablation studies and additional results

C.2.1 Generalized magnitude pruning

In Section 3, we extend the magnitude pruning approach to cater to both NNZ and FLOP constraints. This part is dedicated to investigating the performance of networks pruned using our generalized magnitude pruning approach, denoted as MP-FLOPs. Table 3 analyzes of the test accuracy between MP-FLOPs, MP, and `FALCON` across three networks: ResNet20, MobileNetV1, and ResNet50. As shown in the table, MP-FLOPs consistently perform better than MP in all the tested scenarios. This superior performance results from taking into account both FLOPs and sparsity constraints, which allows MP-FLOPs to consider models with varying FLOPs, NNZ, and accuracy trade-offs, leading to enhanced performance. On the other hand, there remains a significant gap in performance between MP-FLOPs and our proposed model `FALCON`. This is because our proposed optimization framework considers gradient and Hessian information of the loss function, which allows us to identify a better support of the weight vector refine the weights on the support. This experiment validates the effectiveness of our proposed optimization model `FALCON`.

C.2.2 Sparsity of pruned models

In Table 4, we present the percentages of NNZs for ResNet20, MobileNetV1, and ResNet50 that have been pruned using different methods. By simultaneously accounting for both FLOP and NNZ constraints, `FALCON` prunes the network that strikes the optimal balance between FLOPs and NNZs. This leads to networks with slightly more NNZs, yet significantly higher accuracy compared to other approaches.

C.2.3 Additional comparison with other pruning method

We conduct an additional one-shot pruning comparison between `FALCON` and CAIE Wu et al. (2020). Unlike `FALCON` and other competitors, CAIE is a structured pruning approach that prunes entire filters in a convolutional network, not just individual weights. Nonetheless, CAIE's methodology is adaptable to unstructured pruning scenarios. For a fair comparison, we implemented CAIE's algorithm in an unstructured pruning setting, assessing CAIE's loss impact using a second-order Taylor approximation and calibrating its NNZ budget to optimize accuracy. The results, presented in Table 5, demonstrate that `FALCON` consistently outperforms CAIE in terms of accuracy. This superior performance can be attributed to the efficacy of our proposed optimization method.

Table 3: The pruning performance (accuracy) of various methods on ResNet20, MobileNetV1, and ResNet50. The bracketed number in the FLOPs column indicates the proportion of FLOPs needed for inference in the pruned network versus the dense network.

Network	FLOPs	MP	MP-FLOPs	FALCON
ResNet20 on CIFAR10 (91.36%)	24.3M (60%)	88.89	89.49	91.38
	20.3M (50%)	85.95	88.36	90.87
	16.2M (40%)	80.42	84.53	89.67
	12.2M (30%)	58.78	71.07	84.42
	8.1M (20%)	15.04	27.20	65.17
	4.1M (10%)	10.27	18.90	19.14
MobileNetV1 on ImageNet (71.95%)	398M (70%)	70.89	71.03	71.83
	341M (60%)	67.34	67.34	71.42
	284M (50%)	51.13	55.20	70.35
	227M (40%)	14.85	21.36	67.18
	170M (30%)	0.65	1.53	58.40
	113M (20%)	0.10	0.14	25.82
ResNet50 on ImageNet (77.01%)	2.3G (60%)	75.38	75.56	76.86
	2.0G (50%)	70.85	72.82	76.44
	1.6G (40%)	62.37	64.35	75.28
	1.2G (30%)	22.43	38.35	71.54
	817M (20%)	0.56	1.55	54.27
	408M (10%)	0.10	0.11	0.46

Table 4: The percetage of NNZs in ResNet20, MobileNetV1, and ResNet50 pruned by various methods. The bracketed number in the FLOPs column indicates the proportion of FLOPs needed for inference in the pruned network versus the dense network.

Network	FLOPs	MP	WF	CHITA	FALCON
ResNet20 on CIFAR10 (91.36%)	24.3M (60%)	53.9%	59.3%	53.9%	60.0%
	20.3M (50%)	43.1%	49.1%	43.1%	53.0%
	16.2M (40%)	32.6%	39.1%	32.6%	46.0%
	12.2M (30%)	22.7%	29.1%	22.7%	33.0%
	8.1M (20%)	13.5%	19.1%	13.5%	23.0%
	4.1M (10%)	5.2%	9.0%	5.2%	11.0%
MobileNetV1 on ImageNet (71.95%)	398M (70%)	66.3%	71.1%	66.3%	69.0%
	341M (60%)	55.6%	63.1%	55.6%	59.0%
	284M (50%)	44.9%	54.8%	44.9%	52.0%
	227M (40%)	34.1%	45.9%	34.1%	45.0%
	170M (30%)	23.5%	36.5%	23.5%	31.0%
	113M (20%)	13.4%	26.2%	13.4%	21.0%
ResNet50 on ImageNet (77.01%)	2.3G (60%)	56.2%	64.4%	56.2%	65.0%
	2.0G (50%)	44.9%	54.6%	44.9%	55.0%
	1.6G (40%)	33.6%	44.4%	33.6%	48.0%
	1.2G (30%)	22.8%	33.8%	22.8%	38.0%
	817M (20%)	12.7%	22.8%	12.7%	25.0%
	408M (10%)	4.4%	11.4%	4.4%	15.0%

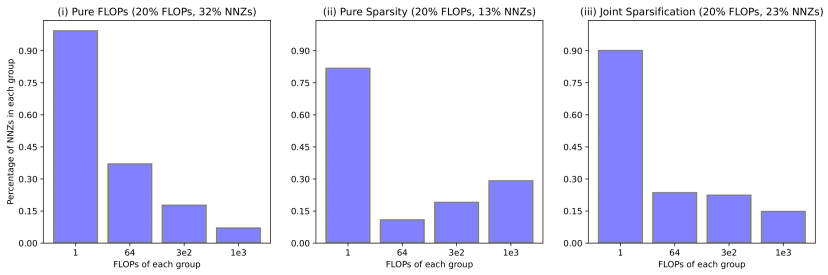
Network	FLOPs	CAIE	FALCON	Network	FLOPs	CAIE	FALCON	Network	FLOPs	CAIE	FALCON
ResNet20 on CIFAR10 (91.36%)	24.3M (60%)	89.54	91.38	MobileNetV1 on ImageNet (71.95%)	398M (70%)	70.48	71.83	ResNet50 on ImageNet (77.01%)	2.3G (60%)	75.51	76.86
	20.3M (50%)	87.33	90.87		341M (60%)	66.72	71.42		2.0G (50%)	72.98	76.39
	16.2M (40%)	80.83	89.67		284M (50%)	54.12	70.35		1.6G (40%)	63.34	75.28
	12.2M (30%)	52.44	84.42		227M (40%)	17.01	67.18		1.2G (30%)	29.41	71.54
	8.1M (20%)	23.18	65.17		170M (30%)	1.25	58.40		817M (20%)	0.75	54.27
	4.1M (10%)	15.13	19.14		113M (20%)	0.12	25.82		408M (10%)	0.10	0.46

Table 5: The accuracy comparison between CAIE and FALCON on ResNet20, MobileNetV1, and ResNet50.

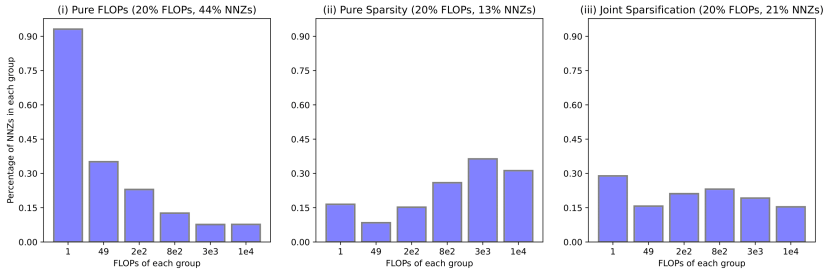
C.2.4 Sparsity distribution of models pruned by FALCON

In this part, we assess the sparsity distribution of models pruned using FALCON within a fixed FLOP budget (F_0). We explore three distinct scenarios: (i) pure FLOP constraint: we set the FLOP budget to $F = F_0$ and the NNZ budget to $S = \infty$; (ii) pure sparsity constraint: we fix $F = \infty$ and find the NNZ budget S so that the FLOPs of the resulting network precisely equals F_0; (iii) joint sparsification: we choose $F = F_0$ and S optimally to maximize accuracy.

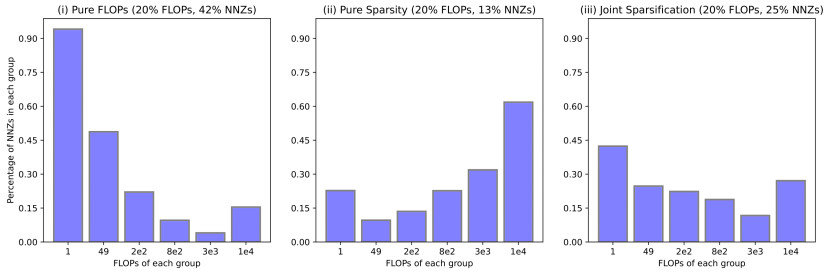
In Figure 4, we display the group-wise sparsity in Resnet20, MobileNetV1, and Resnet50 models pruned by FALCON in these three scenarios. Our findings confirm that joint sparsification achieves more balanced sparsity across groups. This substantiates the claim in Section 5.1. And as explained in Section 5.1, this is why joint sparsification achieves higher accuracy than pure FLOPs or sparsity pruning.



(a) Resnet20



(b) MobileNetV1



(c) Resnet50

Figure 4: Sparsity of each group of models pruned under a fixed FLOP budget (20% of total FLOPs) by FALCON across three scenarios: pure FLOP constraint, pure sparsity constraint, and joint sparsification.