BALF: Budgeted Activation-Aware Low-Rank Factorization for Fine-Tuning-Free Model Compression

Anonymous authors
Paper under double-blind review

Abstract

Activation-aware low-rank factorization techniques yield strong compression results but are generally confined to linear layers, while existing whitening-based theory typically makes an implicit full-rank assumption on activations. We introduce a layer representation framework that extends activation-aware factorization beyond linear layers, including standard and grouped convolutions. Within this framework, our whitening-based formulation is more general than prior ones, naturally covering rank-deficient activations, and yields an optimal low-rank projection that attains the reconstruction error of the best low-rank approximation to layer activations. The resulting singular spectrum provides a closed-form per-layer distortion proxy, which we use to allocate per-layer ranks under explicit FLOP or parameter-count budgets via a Lagrangian relaxation with negligible overhead. Together, these components form BALF, an end-to-end pipeline for efficient vision model compression. Across CNNs and vision transformers on CIFAR-10 and ImageNet-1K, BALF generally achieves higher accuracy than SVD-based factorization baselines at matched FLOP or parameter count targets and remains competitive with other fine-tuning-free compression techniques.

1 Introduction

Deep learning models achieve state-of-the-art results across many domains (LeCun et al., 2015), yet their computational and memory demands limit practical deployment, motivating the need for model compression methods. Among these, low-rank factorization is a strong candidate. Traditional techniques use singular value decomposition (SVD)-based methods to minimize the discrepancy between original and compressed parameters (Jaderberg et al., 2014; Hua et al., 2023). Nonetheless, such techniques often rely on fine-tuning and/or costly search procedures.

Recently, the rise of large language models (LLMs) has motivated activation-aware factorization techniques, which seek to reduce the expected output distortion at each layer (Wang et al., 2025c;b; Chen et al., 2021), often reducing the need for fine-tuning. These techniques predominantly focus on fully connected layers and LLMs, which are central to transformers (Vaswani et al., 2017), leaving other layer types and architectures less explored.

Here, we propose a unified activation-aware factorization framework and introduce an efficient rank allocator that, given FLOP or parameter-count budgets, determines the compression ratio of each layer with near-zero overhead. Together, these form BALF, a principled fine-tuning-free compression pipeline for vision models. More concretely, our main contributions and findings are:

- **Unified and principled framework.** We introduce $(\overline{O}, \overline{I}, \overline{P})$-expressible layers, a framework that enables us to generalize whitening-based activation-aware low-rank factorization to a broad class of layers, including both grouped and ungrouped convolutional layers. Under this framework, we generalize the whitening-based activation-aware theory of (Wang et al., 2025c;b). In particular, our framework explicitly handles rank-deficient activations, avoiding the need to resort to ad hoc fixes such as adding diagonal jitter (Wang et al., 2025c), and generalizes the whitening matrix formulation.

- **Budget-aware allocation.** Guided by a closed-form activation distortion measure, we introduce an efficient rank allocator that approximately meets user-specified FLOP or parameter-count budgets by using a Lagrangian relaxation of the combinatorial rank selection problem.
- **Practicality.** BALF runs end-to-end efficiently on commodity hardware; for example, on an RTX 2080 Ti GPU, it takes less than four minutes to compress each model we tested. It also avoids expensive hyperparameter sweeps by directly targeting compression budgets.
- **Empirical performance.** Across vision models from ResNet-20 (He et al., 2016) on CIFAR-10 (Krizhevsky, 2009) to ResNeXt-101 (Xie et al., 2017) and ViT-B/16 (Dosovitskiy et al., 2021) on ImageNet-1K (Deng et al., 2009), BALF generally surpasses SVD-based factorization baselines at matched FLOP or parameter-count objectives and achieves competitive results against other fine-tuning-free compression techniques.

2 Related Work

Deep learning model compression has been extensively studied. The principal categories of compression techniques include quantization, which employs low-bit storage and operations (Jacob et al., 2018; Frantar et al., 2023; Gholami et al., 2021); pruning, which eliminates parameters or parameter groups from the model (Hoefler et al., 2021; Frantar & Alistarh, 2022); and factorization, which decomposes model layers into smaller sub-layers to reduce memory usage and computational demands (Jaderberg et al., 2014; Ben Noach & Goldberg, 2020). This paper focuses on the factorization approach.

Various factorization methods have been studied. Among these, singular value decomposition (SVD) (Golub & Van Loan, 2013) stands out as a straightforward approach. It is directly applicable to fully connected layers, such as the linear projections commonly found in transformer architectures, which are implemented as matrix multiplications.

In convolutional neural networks (CNNs), factorization is usually performed by reshaping parameter tensors into matrices and subsequently decomposing them (Yang et al., 2020). An alternative line of work employs other kinds of tensor decompositions (Denton et al., 2014; Yin et al., 2021; Phan et al., 2020; Liebenwein et al., 2021; Lebedev et al., 2014).

Existing factorization approaches often rely on fine-tuning and/or compute-intensive search (Idelbayev & Carreira-Perpiñán, 2020; Yu & Bouganis, 2023). The difficulty of running such methods on common hardware motivates the need for lighter alternatives. For example, several pruning methods explicitly aim to be efficient in this regard (Narshana et al., 2023; Murti et al., 2023; Chen et al., 2024; Zhang et al., 2024; Wang et al., 2025a), whereas fine-tuning-free factorization methods are less common—for instance, Yu & Bouganis (2023). Liebenwein et al. (2021); Idelbayev & Carreira-Perpiñán (2020) also report results in the fine-tuning-free regime.

Recent factorization research, motivated by the computational demands of LLMs, leverages intermediate model features to identify additional redundancies suitable for decomposition (Chen et al., 2021; Wang et al., 2025c;b; Qinsi et al., 2025; Yuan et al., 2025). These methods use a calibration dataset to characterize the distribution of activation tensors and then use that information to guide parameter decompositions, generally yielding closed-form solutions and often reducing the need for fine-tuning. However, they primarily focus on decomposing fully connected layers in language models, often overlooking other settings such as image classification, which we explore in this work.[1]

Several formulations of activation-aware factorization have been proposed. In particular, SVD-LLM (Wang et al., 2025c;b) frames its solution using activation-based whitening. However, the whitening-based theory in SVD-LLM implicitly assumes full-rank activations. Our whitening formulation is inspired by SVD-LLM but explicitly allows for rank-deficient activations and generalizes the whitening matrix formulation. This limitation is not inherent to activation-aware factorization: other formulations, such as DRONE (Chen et al., 2021), also allow for rank-deficient activations. Relatedly, the field of reduced-rank regression addresses

[1] Zhang et al. (2016) explore a similar data-aware approach in the image classification domain, but they also take nonlinearities into account.

similar problems; see, for example, Reinsel et al. (2022). Other activation-aware extensions have also been proposed, for example by accounting for compression shift in subsequent layers (Sinha & Fleuret, 2026).

Regarding rank selection, approaches range from uniform rank selection (Wang et al., 2025c; Chen et al., 2021) to more elaborate criteria (Abbasi et al., 2026; Wang et al., 2025b; Zhang et al., 2016), including training-based search (Qinsi et al., 2025; Idelbayev & Carreira-Perpiñán, 2020), among many others.

Concurrent with the development of this work, Ha & Jeon (2026) propose a closely related rank-selection approach for activation-aware model compression, but focus on LLMs. In addition, Kalle et al. (2025) enable convolutional layer compression by incorporating activation-aware decomposition, which they refer to as distribution-aware decomposition, into tensor decomposition methods through an iterative rather than closed-form solution.

3 Preliminaries and Notation

The singular value decomposition. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{M \times N}$, we denote its SVD by $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{M \times M}$ and $\boldsymbol{V} \in \mathbb{R}^{N \times N}$ are orthogonal matrices, and $\boldsymbol{\Sigma} = \operatorname{diag}(\boldsymbol{\sigma}; (M, N))$ is a diagonal matrix containing the singular values. We assume the singular values are sorted in decreasing order. We also use the P-truncated SVD, which is defined by retaining only the P largest singular values and their associated singular vectors. In particular, we define $\mathcal{T}_P^{\mathrm{SVD}}(\boldsymbol{A}) = \boldsymbol{U}_{:,:P}\boldsymbol{\Sigma}_{:P,:P}(\boldsymbol{V}_{:,:P})^\top$.

Uncentered whitening. Data whitening (Kessy et al., 2018) (also termed data sphering) is a widely used technique in machine learning and statistics. Here, we are interested in a special case called uncentered whitening. This process transforms data so that its second-moment matrix becomes the identity matrix or, in the rank-deficient case, a rank-R coordinate projector. Formally, we define an uncentered whitening matrix (UWM) $\boldsymbol{M} \in \mathbb{R}^{I \times I}$ for data $\boldsymbol{X} \in \mathbb{R}^{N \times I}$ as a matrix such that $\boldsymbol{M}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{M} = N\overline{\boldsymbol{I}}_R$, where $\overline{\boldsymbol{I}}_R \in \mathbb{R}^{I \times I}$ is the identity on the first R coordinates and zero elsewhere, and R is the rank of $\boldsymbol{X}$. We further restrict the definition by requiring that $\operatorname{row}(\boldsymbol{X}) = \operatorname{range}(\boldsymbol{M})$, which in turn forces the last $I - R$ columns of $\boldsymbol{M}$ to be zero and makes $\boldsymbol{X} = \boldsymbol{X}\boldsymbol{M}\boldsymbol{M}^+$. For a formal discussion, see Appendix B.1.

Other notation and conventions. We refer to general tensors as $\mathbf{X}$. When a tensor is known to be a matrix or a vector, we may use $\boldsymbol{X}$ and $\boldsymbol{x}$, respectively. For matrices, we denote the Moore-Penrose inverse (also termed the pseudoinverse) of $\boldsymbol{M}$ by $\boldsymbol{M}^+$.

Deep learning frameworks such as PyTorch (Ansel et al., 2024) usually include "reshape" and "permute" operators, denoted in this work by $\mathsf{reshape}(\mathbf{X}; (s_1, \ldots, s_n))$ and $\mathsf{permute}(\mathbf{X}; (\pi_1, \ldots, \pi_n))$, respectively.

When referring to network layers, unless otherwise specified, we drop layer identifiers and write the computation carried out by a layer as $f(\mathbf{X}; \mathbf{W})$, where $\mathbf{X}$ denotes the layer input and $\mathbf{W}$ denotes the layer parameters. In the case of the first layer, $\mathbf{X}$ denotes the input to the network (e.g., a multi-channel 2D signal). In general, when discussing factorization techniques, we ignore bias terms because they are unaffected by factorization; only the primary operations (e.g., convolutions or matrix multiplications) are modified.

We denote the identity function by id. Moreover, when discussing convolutional layers, we use B to denote the batch size; H_x, W_x, C_x to denote the height, width, and channels of the input when $x = i$ and output when $x = o$; and H_k, W_k to denote the spatial size of the kernel. In this context, G denotes the number of groups in the convolutional layer (Xie et al., 2017). We denote the convolution operator by $\mathbf{X} * \mathbf{W}$. Moreover, $\|\cdot\|_F$ refers to the tensor Frobenius norm, i.e., $\|\cdot\|_F = \|\mathsf{flatten}(\cdot)\|_2$.

4 Our Framework

Before diving into our factorization scheme, we first restrict our discussion to a particular class of layers that can be expressed as a combination of matrix multiplications and auxiliary functions.

Definition 4.1. A layer f is $(\overline{O}, \overline{I}, \overline{P})$-expressible if it can be expressed as $f(\mathbf{X}; \mathbf{W}) = \overline{O}(\overline{I}(\mathbf{X})\overline{P}(\mathbf{W}))$, where $\overline{O}$ and $\overline{P}$ are compositions of reshape and permute operators, and $\overline{I}$ is a linear operator. $\overline{O}$ takes a batch

of matrices in the form of a third-order tensor and outputs a tensor (possibly a matrix), whereas $\overline{P}$ and $\overline{I}$ both take a tensor (possibly a matrix) and output a batch of matrices in the form of a third-order tensor of compatible shapes. In particular, $\overline{O} : \mathbb{R}^{G\times N\times O} \to \cdot$, $\overline{I} : \cdot \to \mathbb{R}^{G\times N\times I}$, and $\overline{P} : \cdot \to \mathbb{R}^{G\times I\times O}$.

G stands for the number of groups. In the special case where $G = 1$, we may simply refer to matrices instead of batches containing a single matrix. Note that in the preceding definition, matrix multiplication is batched, i.e., G independent matrix multiplications are carried out. The necessity of batch semantics in the group dimension will become apparent later. This condition is satisfied by popular layers. In what follows, layers are assumed to operate in batched mode, with B denoting the batch size. We refer to N as the outer dimension.

Example 4.2. *Fully connected layers are* $(\mathrm{id}, \mathrm{id}, \mathrm{id})$*-expressible.*

In some architectures (e.g., transformers), linear layers receive inputs with multiple leading independent dimensions (for example, a tensor of shape (B, L, D)). In those cases, we flatten the leading dimensions, reshaping the input to (BL, D) and restoring the original shape afterward. We now turn our attention to convolutional layers. Although an ungrouped convolution is a special case of a (possibly grouped) convolution, we present them separately for clarity.

Example 4.3. *Ungrouped convolutional layers are* $(\overline{O}, \overline{I}, \overline{P})$*-expressible, with*

$$\begin{aligned}
\overline{O}(\mathbf{Y}) &= \mathsf{permute}(\mathsf{reshape}(\mathbf{Y}; (B, H_o, W_o, C_o)); (0, 3, 1, 2)),\\
\overline{I}(\mathbf{X}) &= \mathsf{reshape}(\mathsf{im2col}(\mathbf{X}); (BH_oW_o, C_iH_kW_k)),\\
\overline{P}(\mathbf{W}) &= \mathsf{reshape}(\mathbf{W}; (C_o, C_iH_kW_k))^\top,
\end{aligned}$$

where $\mathsf{im2col} : \mathbb{R}^{B\times C_i\times H_i\times W_i} \to \mathbb{R}^{B\times H_oW_o\times C_iH_kW_k}$ maps a batch of 2D signals with C_i channels into a matrix (see Chellapilla et al. (2006)). The PyTorch equivalent is called unfold[2][3]. It encapsulates the convolution hyperparameters—kernel size, stride, padding, and dilation—but not the number of groups. For brevity, we omit the encapsulated hyperparameters from the notation.

On the other hand, a grouped convolution can be viewed as G separate, parallel convolutions. Each one takes a disjoint slice of C_i/G input channels and produces C_o/G output channels. Concatenating the G outputs recovers the full C_o channels (Krizhevsky et al., 2012; Xie et al., 2017). This also fits our framework.

Example 4.4. *Grouped convolutional layers are* $(\overline{O}, \overline{I}, \overline{P})$*-expressible, with*

$$\begin{aligned}
\overline{O}(\mathbf{Y}) &= \mathsf{permute}(\mathsf{reshape}(\mathsf{permute}(\mathbf{Y}; (0, 2, 1)); (C_o, B, H_o, W_o)); (1, 0, 2, 3)),\\
\overline{I}(\mathbf{X}) &= \mathsf{permute}(\mathsf{reshape}(\mathsf{im2col}(\mathbf{X}); (BH_oW_o, G, (C_i/G)H_kW_k)); (1, 0, 2)),\\
\overline{P}(\mathbf{W}) &= \mathsf{permute}(\mathsf{reshape}(\mathbf{W};\ (G, C_o/G, (C_i/G)H_kW_k)); (0, 2, 1)).
\end{aligned}$$

Additionally, Example 4.3 and Example 4.4 have trivial extensions to higher-dimensional convolutions, achieved by replacing $\mathsf{im2col}$ with its higher-dimensional counterparts.

Low-rank factorization applies naturally to linear layers by decomposing their weight matrices. The main idea of our framework is to extend the notion of factorization to $(\overline{O}, \overline{I}, \overline{P})$-expressible layers by exploiting their equivalence to (possibly batched) matrix multiplications.

Let us focus on linear layers for a moment. Recall that a low-rank matrix can be expressed as the product of a tall matrix and a wide matrix. $\mathcal{T}_P(\boldsymbol{W})$ is defined to be a generic operator that returns a low-rank projection of $\boldsymbol{W}$. If we fix P and approximate $\boldsymbol{X}\boldsymbol{W} \approx \boldsymbol{X}\mathcal{T}_P(\boldsymbol{W}) = (\boldsymbol{X}\boldsymbol{W}_0)\boldsymbol{W}_1$, with $\boldsymbol{W} \in \mathbb{R}^{I\times O}, \boldsymbol{W}_0 \in \mathbb{R}^{I\times P}, \boldsymbol{W}_1 \in \mathbb{R}^{P\times O}$, for sufficiently small P, the storage and compute requirements will be less than those of the original layer (Appendix A). By using their $(\overline{O}, \overline{I}, \overline{P})$ representation, we can extend this to, for instance, convolutional layers. It can be verified that ($\overline{P}^{-1}$ denotes the inverse) $\mathbf{X} * \mathbf{W} \approx \mathbf{X} * \overline{P}^{-1}\left(\mathcal{T}_P\left(\overline{P}(\mathbf{W})\right)\right) = (\mathbf{X} * \mathbf{W}_0) * \mathbf{W}_1$ for some $\mathbf{W}_0 \in \mathbb{R}^{P\times C_i\times H_k\times W_k}, \mathbf{W}_1 \in \mathbb{R}^{C_o\times P\times 1\times 1}$, obtained from decomposing $\overline{P}(\mathbf{W})^\top \in \mathbb{R}^{C_o\times C_iH_kW_k}$. Again, with sufficiently small P, this results in storage and compute savings.

[2] https://docs.pytorch.org/docs/stable/generated/torch.nn.Unfold.html

[3] GPTQ (Frantar et al., 2022) employs a similar formulation to handle convolutional layers in their code (https://github.com/IST-DASLab/gptq/blob/main/gptq.py), although they don't cover grouped layers. Similarly, Zhang et al. (2016) roughly describe a similar idea.

SVD-based factorization. SVD-based factorization is simply a specific way of obtaining two low-rank factors from a matrix. By the Eckart–Young–Mirsky (EYM) theorem (Golub & Van Loan, 2013), it is optimal in terms of parameter distortion, defined as

$$\ell^{\text{param}}(\mathbf{W}, P) = \|\mathbf{W} - \mathcal{T}_P^{\text{SVD}}(\mathbf{W})\|_F^2. \tag{1}$$

In order to simplify the notation, we define $\mathcal{T}_P^{\text{SVD}}(\mathbf{W}) \equiv \overline{P}^{-1}\left(\mathcal{T}_P^{\text{SVD}}(\overline{P}(\mathbf{W}))\right)$. In other words, we extend the matrix low-rank projection notion to arbitrary tensors that represent the weights of $(\overline{O}, \overline{I}, \overline{P})$-expressible layers. In the case where $G > 1$, we decompose each group $\overline{P}(\mathbf{W})_g$ separately, with the constraint that the per-group rank is the same across groups. This latter constraint has practical reasons, which are discussed in Appendix A.3.1.

General low-rank projections. Note that there are many ways one can obtain low-rank factors from a matrix (or a tensor, with our defined extension). A general class of operators can be defined (for a group index g) as

$$\mathcal{T}_P^D\left(\overline{P}(\mathbf{W})_g\right) = \arg \min_{\text{rank}(\overline{P}(\widehat{\mathbf{W}})_g) \leq P} D(\overline{P}(\mathbf{W})_g, \overline{P}(\widehat{\mathbf{W}})_g), \tag{2}$$

where D is some penalty function, and $\widehat{\mathbf{W}}$ is the approximate tensor. We use equality to denote an arbitrary choice from the argmin when it is non-unique. Traditional SVD-based decomposition corresponds to the case where D is the Frobenius norm of the (reshaped) parameters, i.e., it penalizes distance in parameter space (see Equation (1)). We use $\mathcal{T}_P^D(\mathbf{W})$ to also denote batched computation, where each group's matrix is decomposed individually. As before, we omit $\overline{P}$ to simplify notation.

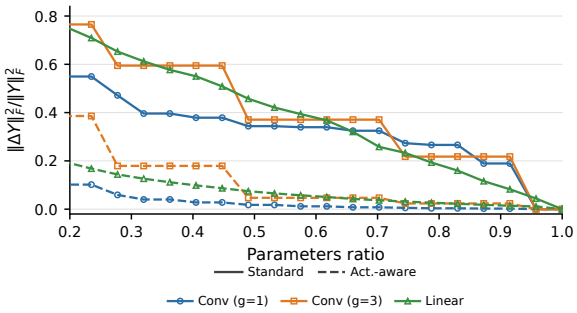

Figure 1: Normalized output distortion (in terms of squared Frobenius norm) with $\mathcal{T}_P^{\text{AA}}(\cdot)$ and $\mathcal{T}_P^{\text{SVD}}(\cdot)$ for different values of P corresponding to different compression ratios, across different layer types. Activation-aware factorization consistently incurs significantly lower distortion.

4.1 Activation-Aware Low-Rank Projections

Motivation. A more natural proxy for task performance is the (expected) distortion in each layer's outputs. In general, though, we have a calibration dataset (which can be the entire training dataset, a subset of it, etc., and is assumed to consist of i.i.d. samples), used to gather layer activations $(\mathbf{X}^{(i)})_{i=1}^B$, which we then use to estimate the distortion. More concretely, the (empirical) average activation distortion for a layer f is defined as

$$\ell^{\text{activ}}(\mathbf{W}, P) = \frac{1}{B}\sum_{i=1}^{B} \|f(\mathbf{X}^{(i)}; \mathbf{W}) - f(\mathbf{X}^{(i)}; \mathcal{T}_P(\mathbf{W}))\|_F^2 = \frac{1}{B}\|f(\mathbf{X}; \mathbf{W}) - f(\mathbf{X}; \mathcal{T}_P(\mathbf{W}))\|_F^2. \tag{3}$$

Note that the last equality holds when forming a "batch" with the list $(\mathbf{X}^{(i)})_{i=1}^B$. In what follows, our analysis is performed in terms of the intermediate activations $\mathbf{X}$ obtained with the calibration dataset, with the expectation that the results carry over to the test dataset. We now aim to find a low-rank projection operator that minimizes Equation (3).

Which projection is best? In order to answer this question, we first need to define what "best" should mean. Without loss of generality, fix a group g. Clearly, $\text{rank}\left(\overline{I}(\mathbf{X})\mathcal{T}_P(\mathbf{W})\right) \leq P$. Moreover, by the EYM theorem, the best rank-P approximation of the output of f in terms of the Frobenius norm is $\mathcal{T}_P^{\text{SVD}}\left(\overline{I}(\mathbf{X})\overline{P}(\mathbf{W})\right)$. Hence, a reasonable notion of an ideal scheme is one that factorizes the parameters in a way that is equivalent to factorizing the outputs of the layer; something better is impossible.

Definition 4.5. We call a scheme $\mathcal{T}_P(\cdot)$ optimal if its distortion is equivalent to directly low-rank truncating the outputs of a layer, i.e., an optimal scheme satisfies

$$\|\overline{I}(\mathbf{X})\overline{P}(\mathbf{W}) - \mathcal{T}_P^{\mathrm{SVD}}\big(\overline{I}(\mathbf{X})\overline{P}(\mathbf{W})\big)\|_F = \|\overline{I}(\mathbf{X})\overline{P}(\mathbf{W}) - \overline{I}(\mathbf{X})\mathcal{T}_P\big(\overline{P}(\mathbf{W})\big)\|_F.$$

Recall that each group is factorized separately. Note that a scheme is optimal if and only if it satisfies Equation (2) with the distance function set to the output distortion. In particular, SVD-LLM (Wang et al., 2025c) introduced a formulation of activation-aware factorization for linear layers, although their formulation implicitly assumes full-rank activations. In practice, they solve the rank-deficient case with diagonal jitter, which is, in general, not optimal[4]. A more general version of their low-rank activation-aware projection method within our framework, which satisfies Definition 4.5 and explicitly allows for rank-deficient activations, can be defined as follows. For any $(\overline{O}, \overline{I}, \overline{P})$-expressible function f, let

$$\mathcal{T}_P^{\mathrm{AA}}(\mathbf{W}) = \overline{P}^{-1}\left(\mathbf{M}\mathcal{T}_P^{\mathrm{SVD}}\left(\mathbf{M}^{+}\overline{P}\left(\mathbf{W}\right)\right)\right),$$

where $\mathbf{M}$ is a batch of whitening matrices for $\overline{I}(\mathbf{X})$, one per group, and the pseudoinverse and matrix multiplications are performed independently across groups.

For the layers and their corresponding $(\overline{O}, \overline{I}, \overline{P})$-tuples mentioned earlier, this corresponds, as discussed in previous examples, to two sequential sub-layers (e.g., two linear or convolutional layers) that yield computational gains with sufficiently low P. More details are given in Appendix A. Additionally, our scheme remains optimal, as claimed in Proposition 4.6.

Proposition 4.6. *$\mathcal{T}_P^{\mathrm{AA}}(\cdot)$ is optimal in the sense of Definition 4.5.*

Moreover, the activation distortion that follows from using our framework is readily computable from the artifacts obtained during compression, as shown in Proposition 4.7.

Proposition 4.7. *Assume that $f(\mathbf{X};\mathbf{W})$ is $(\overline{O}, \overline{I}, \overline{P})$-expressible. Then, the activation distortion (as defined in Equation (3)) incurred when low-rank projecting the parameters with $\mathcal{T}_P^{\mathrm{AA}}(\cdot)$ is $\frac{N}{B}\sum_{g=1}^{G}\sum_{i=P+1}^{U}\boldsymbol{\sigma}_{g,i}^2$, where $\boldsymbol{\sigma}$ denotes the batched singular values of $\mathbf{M}^{+}\overline{P}(\mathbf{W})$ and U is the total number of singular values per group.*

Recall that B is the batch size (number of calibration samples), and N is the outer dimension (see the discussion below Definition 4.1). The main result of Wang et al. (2025c) follows as a corollary.

A simple experiment. We compare the output distortion of our activation-aware projection with that of standard SVD. In particular, we select three layers of different types, truncate them to various rank ratios, and measure the resulting output distortions on 1024 CIFAR-10 test images (using 1024 training images for calibration). As shown in Figure 1, activation-aware factorization yields significantly lower distortion.

4.2 Computing the Whitening Matrices

We compute the whitening matrices using an eigendecomposition. Suppose we are given a group g. Let $\boldsymbol{X} = \overline{I}(\mathbf{X})_g \in \mathbb{R}^{N\times I}$. We can then compute the eigenvectors and eigenvalues of $\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}$: $\boldsymbol{V}$ and $\boldsymbol{\Lambda}$, respectively, and set $\boldsymbol{M} = \boldsymbol{V}\boldsymbol{\Lambda}^{+1/2}$ and $\boldsymbol{M}^{+} = \boldsymbol{\Lambda}^{1/2}\boldsymbol{V}^\top$ (we abuse notation and use $\cdot^{+1/2}$ to denote the square root of the pseudoinverse).

This procedure is applied (in vectorized form) independently for each group. Second moments can be accumulated incrementally over minibatches, so the method is memory-efficient. Additional details are provided in Appendix B.

4.3 Budgeted Rank Allocation

Apart from offering output distortion guarantees, our framework is interpretable in the sense that we can measure the output distortion without additional steps (see Proposition 4.7). Can that be used to efficiently choose the rank to which each layer should be truncated?

[4]In typical implementations, however, we expect no significant empirical impact.

The rank allocation problem. We now turn to the question of how to select the ranks to retain in each layer, $\{P_l\}_{l=1}^{L}$. The naive option is to choose a uniform percentage of rank per layer (Wang et al., 2025c). A more reasonable option, used in the past for the traditional SVD scheme, is the energy-based singular value pruning criterion (Yang et al., 2020; Liebenwein et al., 2021). While more expensive approaches have been explored, such as training-based rank learning (Idelbayev & Carreira-Perpiñán, 2020; Qinsi et al., 2025) or NAS-based search (Yu & Bouganis, 2023), our main goal is to find a scalable and efficient method. We can define the analog of the energy metric in our setting as follows (it reads as the energy retained in layer l when keeping up to rank P_l):

$$E_l(P_l) = \frac{\sum_{g=1}^{G}\sum_{i=1}^{P_l}\boldsymbol{\sigma}_{g,i}^2}{\sum_{g=1}^{G}\sum_{i=1}^{U_l}\boldsymbol{\sigma}_{g,i}^2} = 1 - \frac{\|f_l(\mathbf{X}^{l-1};\mathbf{W}^l) - f_l(\mathbf{X}^{l-1};\mathcal{T}_{P_l}^{\mathrm{AA}}(\mathbf{W}^l))\|_F^2}{\|f_l(\mathbf{X}^{l-1};\mathbf{W}^l)\|_F^2},$$

where l denotes the layer index, U_l the total number of singular values (per group), and P_l the number retained (per group). Energy-based selection then consists of picking the lowest rank such that a certain energy threshold (manually defined by the user) is retained. We propose instead to solve an optimization problem that maximizes the total retained energy globally:

$$\max_{P_1,\ldots,P_L} \sum_{l=1}^{L} E_l(P_l), \quad \text{subject to} \quad \sum_{l=1}^{L} C_l(P_l) \leq C_{\max} \text{ and } P_l \in \{1,\ldots,U_l\},$$

where $C_l(P_l)$ is some measure of layer complexity (in our case, FLOPs or number of parameters) when keeping up to rank P_l on layer l, U_l is the total number of singular values for a group on layer l, and $C_{\max}$ is the global model complexity budget. Our allocator supports FLOPs and the number of parameters as complexity targets. The optimization problem is an instance of the multiple-choice knapsack problem, and it can be formulated as a binary linear program. Unfortunately, it is known to be NP-hard (for a review, see Szkaliczki (2025)), which motivates the need for an alternative.

A Lagrangian relaxation. We approach this limitation by using a Lagrangian relaxation of the problem to obtain approximate solutions. It has a time complexity of $O((I + D + 1)\sum_l U_l)$, where I stands for "number of iterations" and D depends on the spectral properties of the network. In practice, using 300 iterations suffices while taking less than 0.2 seconds, even for the larger models with which we experimented. We provide additional details in Appendix C.1. We note that, concurrent with the development of this work, Ha & Jeon (2026) proposed a very similar idea. Although they discuss it in the context of "water filling", the resulting algorithm is very similar to ours.

Advantages. Our selection approach has several desirable properties: (1) it has virtually zero additional overhead and does not query the model, unlike brute-force search methods, which can take hours (e.g., Yu & Bouganis (2023)), and (2) it provides explicit control over a complexity budget. This reduces the need for manual hyperparameter sweeps to achieve a compression target: a single run is enough.

4.4 Putting It All Together: BALF

Together, these pieces form BALF. An overview can be found in Algorithm 1. The routine NewLayer generates a new layer (consisting of two sequential sub-layers) when given the factorized matrices. In practice, we do not select a manual budget $C_{\max}$, but rather a ratio (e.g., we might want to obtain a model with 50% of its original FLOPs). Additionally, note that we retain the original layer if compressing it to the selected rank is not computationally advantageous. Similar to SVD-LLM, we precompute activations, whitening matrices, and factors once; they are then cached and can be reused. This is significantly more efficient when generating multiple versions of the same model at different compression ratios (see Appendix G for more details about the time cost of each step). One alternative is to compute whitening matrices sequentially, using the outputs of already-compressed preceding layers. Another way is to avoid full recomputation but update relevant elements using distorted inputs, similar to a past SVD-LLM version. In preliminary small-scale experiments, we found that recomputing whitening matrices sequentially offered no significant advantage.

Algorithm 1 BALF overview. Group notation is omitted for simplicity.

Require: Pre-trained model $\{\mathcal{M}_l\}_{l=1}^{L}$, calibration data $\mathcal{D}$, budget $C_{\max}$, cost functions $\{C_l\}_{l=1}^{L}$
Ensure: Compressed model (compressed in place for simplicity)
$\{\frac{1}{N_l}\overline{I}_l(\mathbf{X}^{l-1})^\top \overline{I}_l(\mathbf{X}^{l-1})\}_{l=1}^{L} \leftarrow$ CollectActivations$(\{\mathcal{M}_l\}_{l=1}^{L}, \mathcal{D})$ ▷ stored in reshaped form
$F \leftarrow [\,]$
for $l = 1, \ldots, L$ **do**
  $[\boldsymbol{M}, \boldsymbol{M}^+] \leftarrow$ BatchedUWM$(\frac{1}{N_l}\overline{I}_l(\mathbf{X}^{l-1})^\top \overline{I}_l(\mathbf{X}^{l-1}))$; $\mathbf{W} \leftarrow \mathcal{M}_l.\,\text{weights}$
  $[\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^\top] \leftarrow$ BatchedSVD$(\boldsymbol{M}^+ \overline{P}_l(\mathbf{W}))$
  $F[l] \leftarrow [\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^\top, \boldsymbol{M}, \boldsymbol{M}^+]$
end for
$\{P_l\}_{l=1}^{L} \leftarrow$ AllocateRanks(GetSigmas$(F), C_{\max}, \{C_l\}_{l=1}^{L})$
for $l = 1, \ldots, L$ **do**
  if $C_l(P_l) \geq$ OriginalCost(l) **then continue** ▷ factorizing might not be worth it
  $[\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^\top, \boldsymbol{M}, \boldsymbol{M}^+] \leftarrow F[l]$
  $\boldsymbol{W}_0 \leftarrow \boldsymbol{M}\boldsymbol{U}_{:,:P_l}\boldsymbol{\Sigma}^{1/2}_{:P_l,:P_l}$; $\boldsymbol{W}_1 \leftarrow \boldsymbol{\Sigma}^{1/2}_{:P_l,:P_l}(\boldsymbol{V}^\top)_{:P_l,:}$
  ▷ constructs a new layer composed of two sequential layers depending on the original layer type and hyperparameters
  $\mathcal{M}_l \leftarrow$ NewLayer$(\boldsymbol{W}_0, \boldsymbol{W}_1, \mathcal{M}_l.\,\text{bias}, P_l)$
end for

Handling complex network structures. Our method operates at the layer level: it modifies each layer's internal computation without changing inter-layer connections. As a result, it supports complex architectures (e.g., residual connections) without special handling.

Bounding the output distortion of a model. In general, providing bounds on the cumulative model distortion incurred when compressing it is challenging. In Proposition 4.8, we show a network-level bound on the output distortion of sequential networks in terms of the individual distortions for each layer ℓ_l^{activ}.

Proposition 4.8. *(Informal) Define a sequential L-layer network*

$$\mathbf{X}^l = a_l\left(f_l(\mathbf{X}^{l-1}; \mathbf{W}^l) + \boldsymbol{b}^l\right) = a_l\left(\overline{O}_l\big(\overline{I}_l(\mathbf{X}^{l-1})\overline{P}_l(\mathbf{W}^l)\big) + \boldsymbol{b}^l\right),$$

with $\mathbf{X}^0 = \mathbf{X}$, where a_l is an element-wise activation function with Lipschitz constant A_l. Moreover, let B_l be the Lipschitz constant of $\overline{I}_l$ (recall that $\overline{I}_l$ is linear). We assume that all layers are ungrouped. Then,

$$\frac{1}{\sqrt{B}}\left\|\mathbf{X}^L - \widehat{\mathbf{X}}^L\right\|_F \leq \sum_{l=1}^{L}\Big[A_l\sqrt{\ell_l^{\text{activ}}(P_l)}\prod_{i=l+1}^{L}\|\overline{P}_i(\widehat{\mathbf{W}}^i)\|_2 A_i B_i\Big].$$

A more detailed statement and discussion, along with the proof and special cases, appear in Appendix D. The tightness of the bound depends on the model's depth and its weights, and it can be loose in deep networks. Even so, it remains conceptually informative.

5 Experiments

This section presents our experimental results across a range of architectures, datasets, and scales. Our code is available at `https://anonymous.4open.science/r/BALF-2AFC`. On CIFAR-10 (Krizhevsky, 2009), we evaluate ResNet-20 and ResNet-56, while on ImageNet-1K (Deng et al., 2009), we consider ResNet-18 and ResNet-50 (He et al., 2016), ResNeXt-50 (32×4d) and ResNeXt-101 (32×8d) with grouped convolutions (Xie et al., 2017), ViT-B/16 (Dosovitskiy et al., 2021), DeiT-B/16 (Touvron et al., 2021), and MobileNetV2 (Sandler et al., 2018). For all models, every layer is considered for factorization. Calibration for the main experiments is performed using 1024 images for CIFAR-10 and 8192 images for ImageNet-1K, both uniformly sampled from their respective training datasets. We report accuracy on the CIFAR-10 test set and the ImageNet-1K validation set. The precise settings and hyperparameters are provided in Appendix E.

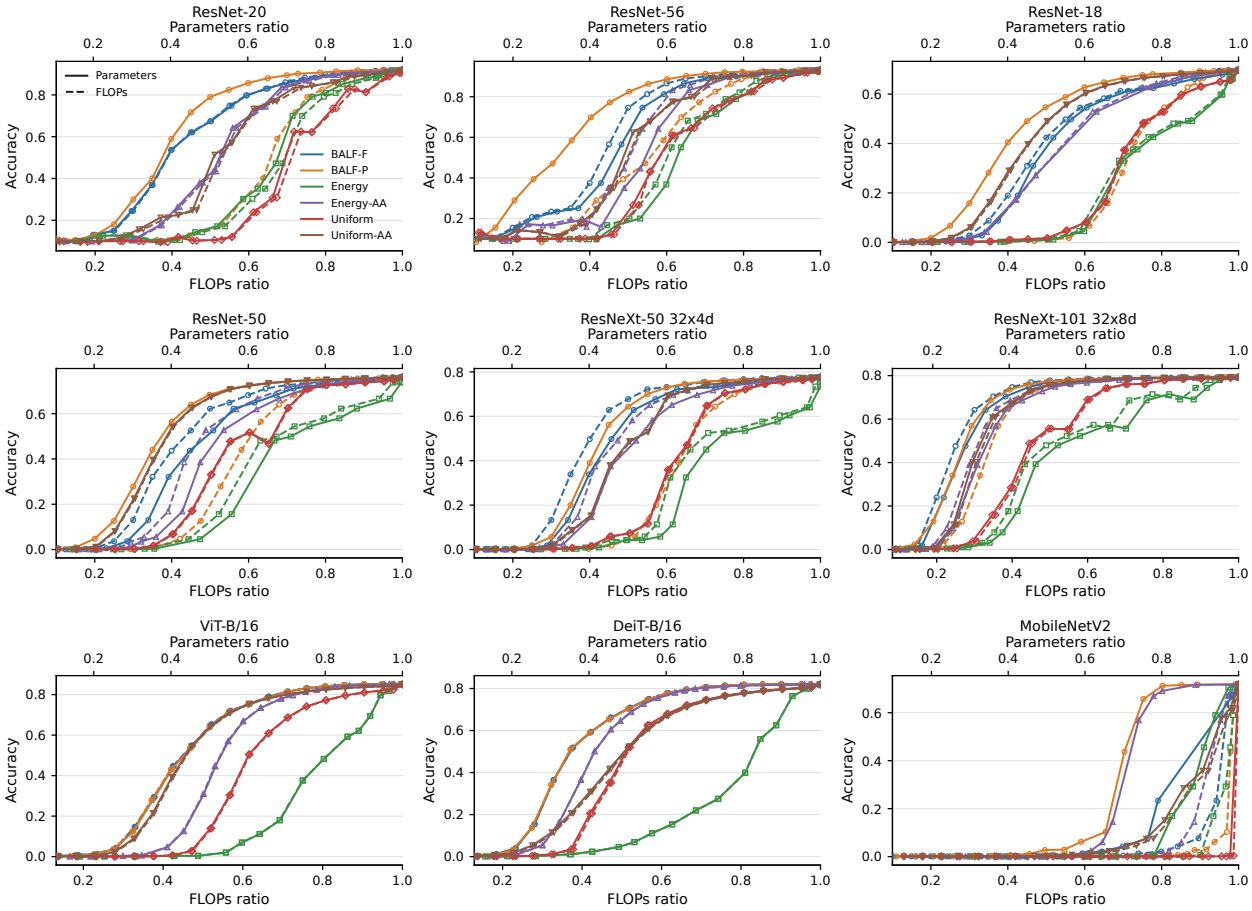

Figure 2: Sweep over compression ratios for different factorization methods across models. Each panel shows parameter–accuracy and FLOP–accuracy curves for each method. BALF-F and BALF-P denote BALF configured with FLOP and parameter-count objectives, respectively. It can be seen that improving the metric targeted by BALF can sometimes incur a nontrivial tradeoff in the non-optimized metric.

Comparison with factorization baselines. We first compare BALF against factorization baselines under matched FLOP and parameter count budgets. We conduct a broad sweep over compression ratios to obtain complexity–accuracy curves measured in FLOPs and parameter counts. To assess the contribution of BALF's components, we compare methods implemented by us: standard SVD and activation-aware SVD, each with both energy-based and uniform rank selection, and BALF, which combines activation-aware decomposition with our rank allocator. For BALF, we consider both FLOP objectives (BALF-F) and parameter-count objectives (BALF-P). For activation-aware methods, which depend on calibration data, we run three calibration seeds and report the mean accuracy. For BALF, whose rank allocation also depends on the calibration data, each reported point averages accuracy and parameter/FLOP counts over the three calibration seeds. In our experiments, the variability induced by the calibration set is negligible, both in terms of compression and accuracy, as can be seen in Tables 1 and 2.

Figure 2 summarizes the complexity–accuracy trade-offs across the different settings. Generally, activation-aware SVD yields consistent gains over plain SVD, with the exact gap varying by configuration. Moreover, our rank allocator enables further low-distortion compression in most cases (but not every case) with respect to the optimized metric (FLOPs or parameter counts). For example, we can reduce the FLOPs of the ResNeXt-101 model by 50% with only about 1.5 percentage points of top-1 accuracy drop. It is important to note, however, that optimizing for one complexity metric can degrade the other. For example, in some cases, setting a parameter-count objective yields substantial gains in parameter count but leads to worse FLOP reduction.

Table 1: Comparison of different methods at different compression configurations on ImageNet-1K models. BALF-P-x and BALF-F-y denote our method when set with parameter count and FLOP objectives, respectively. "-" is used for unavailable results. ΔFLOPs% and ΔParams% denote the percentage changes in FLOPs and parameter counts relative to the original model. For BALF, we sample three seeds and report mean and standard deviation (random variability comes from the calibration subset); results are generally robust to calibration dataset variability.

(a) ResNet-18

Method	ΔFLOPs%	ΔParams%	ΔTop-1 pp
T-ALS	–	−28.57	−10.90
T-ALS	–	−33.33	−17.60
T-ALS-S	–	−28.57	−3.00
T-ALS-S	–	−43.50	−6.50
CP-ALS	–	−51.69	−3.22
CP-ALS	–	−60.47	−8.03
CP-ALS-S	–	−51.69	−1.91
CP-ALS-S	–	−69.23	−6.70
SVD-NAS	−58.60	−68.05	−13.35
ALDS	−42.31	−65.14	−18.70
BALF-F-0.6	$-41.18_{\pm 0.00}$	$-39.45_{\pm 0.00}$	$-15.25_{\pm 0.05}$
BALF-P-0.6	$-13.26_{\pm 0.00}$	$-40.01_{\pm 0.00}$	$-7.00_{\pm 0.02}$
BALF-F-0.7	$-30.71_{\pm 0.01}$	$-28.68_{\pm 0.02}$	$-8.50_{\pm 0.01}$
BALF-P-0.7	$-9.13_{\pm 0.01}$	$-29.99_{\pm 0.01}$	$-3.20_{\pm 0.03}$
BALF-F-0.8	$-20.01_{\pm 0.00}$	$-16.74_{\pm 0.00}$	$-5.49_{\pm 0.04}$
BALF-P-0.8	$-6.77_{\pm 0.01}$	$-22.28_{\pm 0.02}$	$-1.69_{\pm 0.02}$

(b) ResNet-50

Method	ΔFLOPs%	ΔParams%	ΔTop-1 pp
IFM	–	−5.65	−1.46
IFM	–	−20.42	−10.45
ITVSP	–	−4.76	−3.02
ITVSP	–	−9.98	−10.21
DFPC	–	−5.66	−5.78
DFPC	–	−10.92	−13.88
T-ALS	–	−27.54	−4.49
T-ALS	–	−33.33	−20.14
T-ALS-S	–	−27.54	−2.57
T-ALS-S	–	−33.33	−6.90
BALF-F-0.5	$-50.03_{\pm 0.02}$	$-43.84_{\pm 0.01}$	$-13.96_{\pm 0.05}$
BALF-P-0.5	$-28.17_{\pm 0.01}$	$-50.06_{\pm 0.01}$	$-7.79_{\pm 0.05}$
BALF-F-0.7	$-30.01_{\pm 0.00}$	$-21.66_{\pm 0.01}$	$-3.50_{\pm 0.01}$
BALF-P-0.7	$-13.60_{\pm 0.00}$	$-29.95_{\pm 0.00}$	$-1.97_{\pm 0.03}$
BALF-F-0.8	$-20.03_{\pm 0.02}$	$-9.73_{\pm 0.01}$	$-1.69_{\pm 0.02}$
BALF-P-0.8	$-9.03_{\pm 0.01}$	$-21.90_{\pm 0.01}$	$-1.11_{\pm 0.01}$

(c) DeiT-B/16

Method	ΔFLOPs%	ΔParams%	ΔTop-1 pp
GTP-15.3	−13.07	0.00	0.00
GTP-8.8	−50.00	0.00	−3.50
PRACTISE	−16.6	–	−2.5
DC-ViT	−16.6	–	−0.54
BALF-F-0.6	$-38.37_{\pm 0.00}$	$-39.52_{\pm 0.00}$	$-4.10_{\pm 0.03}$
BALF-P-0.6	$-38.71_{\pm 0.00}$	$-39.87_{\pm 0.00}$	$-4.31_{\pm 0.02}$
BALF-F-0.7	$-28.78_{\pm 0.00}$	$-29.64_{\pm 0.00}$	$-1.25_{\pm 0.05}$
BALF-P-0.7	$-29.04_{\pm 0.00}$	$-29.91_{\pm 0.00}$	$-1.34_{\pm 0.02}$
BALF-F-0.8	$-19.22_{\pm 0.00}$	$-19.79_{\pm 0.00}$	$-0.19_{\pm 0.01}$
BALF-P-0.8	$-19.36_{\pm 0.00}$	$-19.94_{\pm 0.00}$	$-0.17_{\pm 0.01}$

(d) MobileNetV2

Method	ΔFLOPs%	ΔParams%	ΔTop-1 pp
SVD-NAS	−15.09	−9.00	−12.54
ALDS	−2.62	−37.61	−16.95
LR-S2	−3.81	−6.24	−17.46
L2+REST	–	−5.00	−4.42
L2+REST	–	−10.00	−18.47
BALF-P-0.7	$-2.07_{\pm 0.01}$	$-29.72_{\pm 0.02}$	$-28.08_{\pm 0.50}$
BALF-P-0.75	$-1.13_{\pm 0.01}$	$-24.78_{\pm 0.02}$	$-6.17_{\pm 0.05}$
BALF-P-0.8	$-0.23_{\pm 0.00}$	$-19.80_{\pm 0.00}$	$-0.61_{\pm 0.03}$
BALF-P-0.9	$-0.12_{\pm 0.00}$	$-10.69_{\pm 0.05}$	$-0.23_{\pm 0.01}$
BALF-F-0.97	$-2.51_{\pm 0.00}$	$-1.55_{\pm 0.00}$	$-4.89_{\pm 0.02}$

For ViTs, the FLOP and parameter-count curves are nearly identical. This is because, ignoring operations outside the linear layers, compressing a linear layer reduces FLOPs by the same amount as it reduces the number of parameters. MobileNetV2 is a notably challenging case for factorization. Activation-aware SVD yields relatively substantial improvements in the parameter-count setting, with the allocator providing a slight additional edge, but FLOP reduction remains difficult.

Comparison with other fine-tuning-free compression approaches. Additionally, we compare BALF with other structured compression approaches in the fine-tuning-free regime. Unless otherwise noted, we take results from their respective papers.

We compare against several pruning and factorization methods, depending on the availability of reported results. These include the pruning methods DFPC (Narshana et al., 2023), IterTVSPrune (ITVSP) (Murti et al., 2023), IFM (Chen et al., 2024), FVRCP (He et al., 2020), DC-ViT (Zhang et al., 2024), and PRACTISE (Wang & Wu, 2023); the factorization methods ALDS (Liebenwein et al., 2021), SVD-NAS (Yu & Bouganis, 2023) and its LR-S2 baseline, and the methods studied in Kalle et al. (2025); the token-merging method GTP (Xu et al., 2024); and the pruning-restoration method L2+REST (Lee et al., 2025). For Kalle et al.

Method	ΔFLOPs%	ΔParams%	ΔTop-1 pp
ALDS	–	−23.71	−1.50
ALDS	–	−30.05	−3.37
ALDS	–	−50.00	−25.22
FVRCP	−19.84	–	−2.5
FVRCP	−29.51	–	−5.83
FVRCP	−49.35	–	−24.86
BALF-F-0.5	$-50.02_{\pm 0.00}$	$-49.53_{\pm 0.08}$	$-24.51_{\pm 0.23}$
BALF-P-0.5	$-24.75_{\pm 0.00}$	$-49.74_{\pm 0.00}$	$-12.94_{\pm 0.26}$
BALF-F-0.7	$-30.47_{\pm 0.19}$	$-29.06_{\pm 0.03}$	$-5.91_{\pm 0.24}$
BALF-P-0.7	$-12.78_{\pm 0.02}$	$-29.91_{\pm 0.05}$	$-2.52_{\pm 0.04}$
BALF-F-0.8	$-20.03_{\pm 0.00}$	$-17.90_{\pm 0.00}$	$-2.25_{\pm 0.05}$
BALF-P-0.8	$-9.11_{\pm 0.05}$	$-21.31_{\pm 0.11}$	$-1.24_{\pm 0.05}$

Table 2: ResNet-20 on CIFAR-10 results, following Table 1

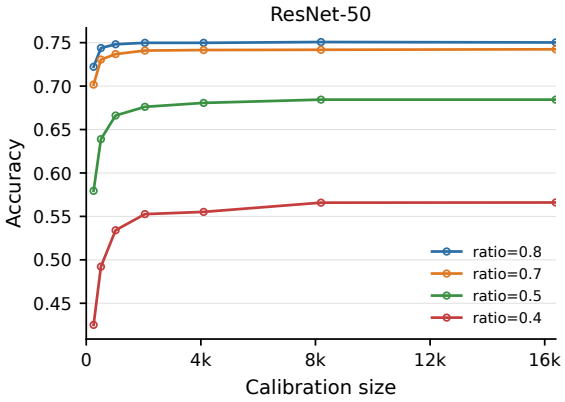

Figure 3: Effect of calibration dataset size on ResNet-50 compression.

(2025), T-ALS and CP-ALS denote the plain alternating least-squares variants, while the -S variants denote their data-aware counterparts.

We take results from their respective papers, with the following exceptions. DeiT-B/16 results for PRACTISE are those reported by Zhang et al. (2024), and the no-fine-tuning MobileNetV2 results for ALDS are those reported by Yu & Bouganis (2023). For L2+REST, which reports the number of removed filters, we approximate this quantity as the number of removed parameters. Note that SVD-NAS requires an expensive search procedure on the order of hours. Table 1 summarizes the results for ImageNet-1K models, and Table 2 includes the results for ResNet-20 on CIFAR-10. Overall, BALF achieves performance comparable to most of the methods considered. It outperforms existing methods in some situations, although it is outperformed in others.

Compression runtime. On an RTX 2080 Ti GPU, compressing ImageNet-1K models is completed in minutes: under four minutes for all models we tested. The most expensive steps can be cached, reducing subsequent compression runs to seconds. We provide a comprehensive benchmark and implementation details in Appendix G.

Robustness under calibration–evaluation mismatch. We evaluate the robustness of BALF under distribution shift in the calibration data. We do so by sampling calibration data from shifted datasets: CIFAR-10-C (Hendrycks & Dietterich, 2019) for ResNet-20, and ImageNet-C (Hendrycks & Dietterich, 2019) and ImageNet-V2 (Recht et al., 2021) for ResNet-18. We then evaluate the compressed models as usual on the original evaluation datasets. We find that results vary depending on the setting. On ImageNet-C and CIFAR-10-C, the effect generally depends on the corruption type and intensity: some corruptions produce only marginal decreases in performance, while more extreme corruptions can substantially affect BALF. In contrast, ImageNet-V2 calibration data yields no degradation, suggesting that BALF is robust to natural calibration shifts.

Robustness under calibration dataset size. We evaluate BALF using different calibration dataset sizes. We find that fewer calibration samples often yield only marginally worse results than those reported in the main experiments (about 256 samples often suffice for CIFAR-10, while about 1024–2048 samples often suffice for ImageNet-1K), depending on the setting. For instance, Figure 3 shows a sweep on ResNet-50. Other results, following similar trends, are presented in Appendix F.

Limitations and future work. BALF inherits some limitations from low-rank factorization. For example, depthwise convolutional layers cannot be further compressed within our framework because the only available rank is $P = 1$. More generally, the $(\overline{O}, \overline{I}, \overline{P})$ framework may not accommodate every layer type. In addition, although BALF reduces theoretical FLOPs and parameter counts, realizing end-to-end speedups can require

efficient low-rank operators; for some configurations, taking advantage of the resulting low-rank structure remains challenging in practice (see Appendix F.3).

Our rank allocator is also approximate. In particular, it can undershoot the target budget in specific cases (see Appendix C.1). Although we did not observe this behavior in our experiments, it can occur theoretically for networks with very sharp spectra, for example as a result of low-rank regularization (Ghosh et al., 2026). Extending the allocator to support additional constraint types and concurrent constraints is an interesting direction for future work. Another possible direction is to combine activation-aware projections with more expensive search methods, such as SVD-NAS, which could improve accuracy at the expense of compression time.

Finally, while BALF performs well on the vision models considered in this work, preliminary experiments on LLMs showed degradation relative to reported results from state-of-the-art factorization methods such as SVD-LLM. Since SVD-LLM-like projections are nearly equivalent to ours in practice, this suggests that our rank allocator may not be suitable in all settings.

Conclusion. We introduced BALF, a fine-tuning-free compression pipeline that generalizes activation-aware low-rank factorization beyond linear layers through our $(\overline{O}, \overline{I}, \overline{P})$-expressible layer framework. Our theory yields an optimal activation-aware projection even under rank-deficient activations, avoiding ad hoc fixes and providing a closed-form distortion proxy for fast, budgeted rank allocation under explicit FLOP or parameter-count constraints. Empirically, BALF consistently improves the accuracy–complexity trade-off over SVD-based baselines across CNNs and vision transformers, generally achieves competitive performance against other fine-tuning-free compression methods, and runs efficiently, compressing models in minutes on commodity GPUs.

References

Ali Abbasi, Chayne Thrash, Haoran Qin, Shansita Sharma, Sepehr Seifi, and Soheil Kolouri. Zero Sum SVD: Balancing Loss Sensitivity for Low Rank LLM Compression, 2026. URL https://arxiv.org/abs/2602.02848.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024. doi: 10.1145/3620665.3640366. URL https://docs.pytorch.org/assets/pytorch2-2.pdf.

Matan Ben Noach and Yoav Goldberg. Compressing Pre-trained Language Models by Matrix Decomposition. In Kam-Fai Wong, Kevin Knight, and Hua Wu (eds.), *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pp. 884–889, Suzhou, China, December 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.aacl-main.88. URL https://aclanthology.org/2020.aacl-main.88/.

Kumar Chellapilla, Sidd Puri, and Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. In Guy Lorette (ed.), *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), October 2006. Université de Rennes 1, Suvisoft. URL https://inria.hal.science/inria-00112631.

Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. DRONE: Data-aware Low-rank Compression for Large NLP Models. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman

Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 29321–29334. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f56de5ef149cf0aedcc8f4797031e229-Paper.pdf.

Yiting Chen, Zhanpeng Zhou, and Junchi Yan. Going Beyond Neural Network Feature Similarity: The Network Feature Complexity and Its Interpretation Using Category Theory. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4bSQ3lsfEV.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval Networks: Improving Robustness to Adversarial Examples. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 854–863. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/cisse17a.html.

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*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pp. 1269–1277, Cambridge, MA, USA, 2014. MIT Press.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Iain Dunning, Stuart Mitchell, and Michael O'Sullivan. PuLP: A Linear Programming Toolkit for Python. Technical report, Department of Engineering Science, The University of Auckland, September 2011. URL https://optimization-online.org/?p=11731.

Elias Frantar and Dan Alistarh. Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=ksVGCOlOEba.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-training Compression for Generative Pretrained Transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=tcbBPnfwxS.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. A Survey of Quantization Methods for Efficient Neural Network Inference. *CoRR*, abs/2103.13630, 2021. URL https://arxiv.org/abs/2103.13630.

Ipsita Ghosh, Ethan Nguyen, and Christian Kümmerle. Q3R: Quadratic Reweighted Rank Regularizer for Effective Low-Rank Training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=ZtzWvNKOCr.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA, 2013. doi: 10.1137/1.9781421407944. URL https://epubs.siam.org/doi/abs/10.1137/1.9781421407944.

Sihyeon Ha and Yo-Seb Jeon. AFORA: Activation-aware Factorization with Optimal Rank Allocation for Training-free LLM Compression, 2026. URL https://openreview.net/forum?id=W5kV18hrYO.

Junjie He, Bohua Chen, Yinzhang Ding, and Dongxiao Li. Feature Variance Ratio-Guided Channel Pruning for Deep Convolutional Network Acceleration. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, November 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 (CVPR)*, June 2016.

Dan Hendrycks and Thomas Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJz6tiCqYm.

Michael Heroux, Ahmad Abdelfattah, Natalie Beams, Robert Carson, Pieter Ghysels, Tzanio Kolev, Thomas Stitt, Arturo Vargas, Stanimire Tomov, and Jack Dongarra. MAGMA: Enabling exascale performance with accelerated BLAS and LAPACK for diverse GPU architectures. *Int. J. High Perform. Comput. Appl.*, 38(5):468–490, September 2024. ISSN 1094-3420. doi: 10.1177/10943420241261960. URL https://doi.org/10.1177/10943420241261960.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021. URL http://jmlr.org/papers/v22/21-0366.html.

Ting Hua, Xiao Li, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Dynamic Low-rank Estimation for Transformer-based Language Models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9275–9287, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.621. URL https://aclanthology.org/2023.findings-emnlp.621/.

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán. Low-Rank Compression of Neural Nets: Learning the Rank of Each Layer. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8046–8056, 2020. doi: 10.1109/CVPR42600.2020.00807.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ioffe15.html.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. doi: http://dx.doi.org/10.5244/C.28.88.

Alper Kalle, Théo Rudkiewicz, Mohamed Ouerfelli, and Mohamed Tamaazousti. Distribution-Aware Tensor Decomposition for Compression of Convolutional Neural Networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ODgWBaErst.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, Berlin, Heidelberg, 1 edition, 2004. ISBN 978-3-540-24777-7. doi: 10.1007/978-3-540-24777-7.

Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal Whitening and Decorrelation. *The American Statistician*, 72(4):309–314, 2018. doi: 10.1080/00031305.2016.1277159. URL https://doi.org/10.1080/00031305.2016.1277159.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, I. Oseledets, and Victor S. Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. *CoRR*, abs/1412.6553, 2014. URL https://api.semanticscholar.org/CorpusID:15002492.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL https://doi.org/10.1038/nature14539.

Keonho Lee, Minsoo Kim, and Dong-Wan Choi. Training-Free Restoration of Pruned Neural Networks, 2025. URL https://arxiv.org/abs/2502.08474.

Lucas Liebenwein, Alaa Maalouf, Dan Feldman, and Daniela Rus. Compressing Neural Networks: Towards Determining the Optimal Layer-wise Decomposition. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=BvJkwMhyInm.

Chaitanya Murti, Tanay Narshana, and Chiranjib Bhattacharyya. TVSPrune - Pruning Non-discriminative filters via Total Variation separability of intermediate representations without fine tuning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=sZI1Oj9KBKy.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Tanay Narshana, Chaitanya Murti, and Chiranjib Bhattacharyya. DFPC: Data flow driven pruning of coupled channels without data. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=mhnHqRqcjYU.

NVIDIA. *cuSOLVER Library*, 2025. URL https://docs.nvidia.com/cuda/cusolver/. Release 13.0.

Anh-Huy Phan, Konstantin Sobolev, Konstantin Sozykin, Dmitry Ermilov, Julia Gusak, Petr Tichavský, Valeriy Glukhov, Ivan Oseledets, and Andrzej Cichocki. Stable Low-Rank Tensor Decomposition for Compression of Convolutional Neural Network. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, pp. 522–539, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58526-6.

Wang Qinsi, Jinghan Ke, Masayoshi Tomizuka, Kurt Keutzer, and Chenfeng Xu. Dobi-SVD: Differentiable SVD for LLM Compression and Some New Perspectives. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kws76i5XB8.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet Classifiers Generalize to ImageNet? In *NeurIPS 2021 Workshop on ImageNet: Past, Present, and Future*, 2021. URL https://openreview.net/forum?id=cwj2tCVncS.

Gregory C. Reinsel, Raja P. Velu, and Kun Chen. *Multivariate Reduced-Rank Regression: Theory, Methods and Applications*, volume 225 of *Lecture Notes in Statistics*. Springer, New York, NY, 2nd edition, 2022. ISBN 978-1-0716-2793-8. doi: 10.1007/978-1-0716-2793-8.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Zakhar Shumaylov, Vasileios Tsiaras, and Yannis Stylianou. On Information Geometry and Iterative Optimization in Model Compression: Operator Factorization, 2025. URL https://arxiv.org/abs/2507.09428.

Atul Kumar Sinha and François Fleuret. AA-SVD: Anchored and Adaptive SVD for Large Language Model Compression, 2026. URL https://arxiv.org/abs/2604.02119.

Yang Sui, Miao Yin, Yu Gong, Jinqi Xiao, Huy Phan, and Bo Yuan. ELRT: Efficient Low-Rank Training for Compact Convolutional Neural Networks, 2024. URL https://arxiv.org/abs/2401.10341.

Tibor Szkaliczki. Solution Methods for the Multiple-Choice Knapsack Problem and Their Applications. *Mathematics*, 13(7), 2025. ISSN 2227-7390. doi: 10.3390/math13071097. URL https://www.mdpi.com/2227-7390/13/7/1097.

TorchVision maintainers and contributors. TorchVision: PyTorch's Computer Vision library, November 2016. URL https://github.com/pytorch/vision.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/touvron21a.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Dong Wang, Haris Šikić, Lothar Thiele, and Olga Saukh. Forget the Data and Fine-Tuning! Just Fold the Network to Compress. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=W2Wkp9MQsF.

Guo-Hua Wang and Jianxin Wu. Practical Network Acceleration With Tiny Sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 20331–20340, June 2023.

Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. SVD-LLM V2: Optimizing Singular Value Truncation for Large Language Model Compression. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4287–4296, Albuquerque, New Mexico, April 2025b. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.217. URL https://aclanthology.org/2025.naacl-long.217/.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware Singular Value Decomposition for Large Language Model Compression. In *The Thirteenth International Conference on Learning Representations*, 2025c. URL https://openreview.net/forum?id=LNYIUouhdt.

Ross Wightman. PyTorch Image Models, 2019. URL https://github.com/huggingface/pytorch-image-models.

Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Xuwei Xu, Sen Wang, Yudong Chen, Yanping Zheng, Zhewei Wei, and Jiajun Liu. GTP-ViT: Efficient Vision Transformers via Graph-Based Token Propagation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 86–95, January 2024.

Huanrui Yang, Minxue Tang, Wei Wen, Feng Yan, Daniel Hu, Ang Li, Hai Li, and Yiran Chen. Learning Low-rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification. *CoRR*, abs/2004.09031, 2020. URL https://arxiv.org/abs/2004.09031.

Miao Yin, Yang Sui, Siyu Liao, and Bo Yuan. Towards Efficient Tensor Decomposition-Based DNN Model Compression With Optimization Framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10683, June 2021.

Zhewen Yu and Christos-Savvas Bouganis. SVD-NAS: Coupling Low-Rank Approximation and Neural Architecture Search. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 1503–1512, January 2023.

Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. ASVD: Activation-aware Singular Value Decomposition for Compressing Large Language Models, 2025. URL https://arxiv.org/abs/2312.05821.

Hanxiao Zhang, Yifan Zhou, and Guo-Hua Wang. Dense Vision Transformer Compression with Few Samples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15825–15834, June 2024.

Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating Very Deep Convolutional Networks for Classification and Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(10):1943–1955, October 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2015.2502579. URL https://doi.org/10.1109/TPAMI.2015.2502579.

Appendix Overview

This appendix is organized as follows:

- Appendix A expands on the factorization scheme and provides parameter and FLOP counts for layers before and after low-rank factorization. It includes results for fully connected, ungrouped convolutional, and grouped convolutional layers, and motivates the use of a shared per-group rank for the latter.
- Appendix B details the construction of uncentered whitening matrices via eigendecomposition, discusses numerical choices and fallbacks, and describes the incremental estimation of second moments and other details.
- Appendix C discusses the specifics of our rank allocator: it specifies the cost model, and presents the previously mentioned Lagrangian relaxation, together with complexity analysis and experimental results on its behavior.
- Appendix D states and proves the formal counterpart of Proposition 4.8 (discussed in the main text), including simplified forms for special cases.
- Appendix E documents the experimental details of our main experiments, including datasets, models, preprocessing, compression protocols, and other important details.
- Appendix F includes additional experiments that answer secondary questions, including reports on calibration dataset size ablations, robustness under distribution shift, and practical speedups.
- Appendix G describes the end-to-end practical implementation of BALF at a high level and provides timing and peak memory measurements of its different parts.
- Appendix H provides a brief analysis of BALF through the lens of Shumaylov et al. (2025).
- Appendix I collects complete proofs for results deferred from the main text, namely Propositions 4.6 and 4.7 and auxiliary lemmas.

A Decomposing the Layers and Complexity After Factorization

In this section, we provide a more thorough discussion of how layers are decomposed. We also provide the parameter counts and FLOP counts before and after decomposition (the latter is used as the cost function in our rank allocator). Given a layer and a rank P, our scheme has the same complexity as basic SVD truncation to rank P; the difference lies in the (expected) quality of the outputs of the factorized layer. We omit biases for simplicity, as these remain constant in the original and factorized layers. B denotes the batch size. Strictly speaking, the FLOP counts reported below should all be doubled (as it stands, we count each multiply-accumulate as one FLOP), but we omit the doubling factor for readability.

A.1 Fully Connected Layers

Let the weight be $\boldsymbol{W} \in \mathbb{R}^{D_i \times D_o}$. For $\boldsymbol{X} \in \mathbb{R}^{B \times D_i}$, the original layer computes

$$f(\boldsymbol{X}; \boldsymbol{W}) = \boldsymbol{X}\boldsymbol{W}.$$

It has a total of $D_i D_o$ parameters, and takes BD_iD_o FLOPs.

After decomposition. Given a rank P, we factor the weights into $\boldsymbol{W}_0 \in \mathbb{R}^{D_i \times P}$ and $\boldsymbol{W}_1 \in \mathbb{R}^{P \times D_o}$. The decomposed layer becomes

$$f(\boldsymbol{X}; \boldsymbol{W}_0, \boldsymbol{W}_1) = (\boldsymbol{X}\boldsymbol{W}_0)\boldsymbol{W}_1.$$

The total storage needed is the sum of the storage for each matrix, that is, $P(D_i + D_o)$, while the number of FLOPs is $BP(D_i + D_o)$ (the computation order is algebraically irrelevant, but the order is important for efficiency).

A.2 2D Convolutional Layers

Let the kernel be $\mathbf{W} \in \mathbb{R}^{C_o \times C_i \times H_k \times W_k}$ and the (per-item) output spatial size be $H_o \times W_o$. For an input $\mathbf{X} \in \mathbb{R}^{B \times C_i \times H_i \times W_i}$, the original layer computes

$$f(\mathbf{X}; \mathbf{W}) = \mathbf{X} * \mathbf{W},$$

where $*$ denotes 2D convolution. The total number of parameters is $C_o C_i H_k W_k$, and the number of FLOPs is $B H_o W_o C_o C_i H_k W_k$.

After decomposition. Choose a rank P. We factor the kernel into

$$\mathbf{W}_0 \in \mathbb{R}^{P \times C_i \times H_k \times W_k} \quad \text{and} \quad \mathbf{W}_1 \in \mathbb{R}^{C_o \times P \times 1 \times 1},$$

and the layer computes

$$f(\mathbf{X}; \mathbf{W}_0, \mathbf{W}_1) = \big(\mathbf{X} * \mathbf{W}_0\big) * \mathbf{W}_1.$$

The total number of parameters is $P(C_i H_k W_k + C_o)$, and the FLOPs are $B H_o W_o P(C_i H_k W_k + C_o)$. Note that H_o and W_o subsume the additional convolution hyperparameters (padding, strides, dilation, etc.).

"Regular" convolutions are a special case of grouped convolutions, but we included a separate discussion for the convenience of the reader. We will now discuss the general case.

A.3 Grouped 2D Convolutional Layers

Let G be the number of groups (assume C_i and C_o are divisible by G, as is generally required in modern frameworks). The grouped kernel has shape

$$\mathbf{W} \in \mathbb{R}^{C_o \times C_i/G \times H_k \times W_k}.$$

The original grouped convolution computes

$$f(\mathbf{X}; \mathbf{W}) = \mathbf{X} \ *_G \ \mathbf{W},$$

with a total of $C_o \frac{C_i}{G} H_k W_k$ parameters, and a total of $B H_o W_o C_o \frac{C_i}{G} H_k W_k$ FLOPs.

After decomposition. Given the per-group rank P, we decompose the weights into

$$\mathbf{W}_0 \in \mathbb{R}^{PG \times C_i/G \times H_k \times W_k} \quad \text{and} \quad \mathbf{W}_1 \in \mathbb{R}^{C_o \times P \times 1 \times 1},$$

and the layer computes two grouped convolutions in sequence:

$$f(\mathbf{X}; \mathbf{W}_0, \mathbf{W}_1) = \big(\mathbf{X} \ *_G \ \mathbf{W}_0\big) \ *_G \ \mathbf{W}_1.$$

The total parameter storage across all groups is $P(C_i H_k W_k + C_o)$, and the FLOPs are $B H_o W_o P(C_i H_k W_k + C_o)$.

The intermediate feature map has PG channels in total—P per group—but the parameter and FLOP counts simplify to the expressions above.

A.3.1 On the Sharing of Group Ranks

As noted in the preceding analysis and in the main text, each group shares the same rank P. This is done for two main reasons.

First, as noted above, the decomposition with the same rank per group results in two sequential (grouped) convolutions. Having different ranks per group would mean that, implementation-wise, we would need G separate convolutions (or at least some kind of bucketing scheme). In practice, this would be very challenging to perform efficiently.

The second reason is that, even if we managed to achieve an optimal implementation, grouped convolutions are, in essence, G smaller convolutions run in parallel. It is expected that the largest group would generally be a bottleneck for the others, as the smaller groups would have to wait for them.

B Uncentered Whitening Matrices

B.1 Formal Definition and Properties

We first give the exact definition used throughout the text.

Definition B.1. We define an uncentered whitening matrix (UWM) $\boldsymbol{M} \in \mathbb{R}^{I \times I}$ for the data $\boldsymbol{X} \in \mathbb{R}^{N \times I}$ as a matrix such that $\boldsymbol{M}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{M} = N \overline{\boldsymbol{I}}_R$, where $\overline{\boldsymbol{I}}_R \in \mathbb{R}^{I \times I}$ is the identity on the first R coordinates and zero elsewhere, and R is the rank of $\boldsymbol{X}$. We further restrict the definition by requiring that $\text{row}(\boldsymbol{X}) = \text{range}(\boldsymbol{M})$.

The following two properties hold, with notation as in Definition B.1.

Proposition B.2. *The last $I - R$ columns of a UWM $\boldsymbol{M}$ are zero.*

Proof. By our definition, $\boldsymbol{M}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{M} = N \overline{\boldsymbol{I}}_R$, which means that the last $I - R$ columns of $\boldsymbol{X}\boldsymbol{M}$ are zero, i.e., the last $I - R$ columns of $\boldsymbol{M}$ lie in $\ker(\boldsymbol{X})$. These columns also lie in $\text{row}(\boldsymbol{X})$. Since $\ker(\boldsymbol{X}) \cap \text{row}(\boldsymbol{X}) = \{0\}$, these columns must be zero. □

Proposition B.3. *It holds that $\boldsymbol{X} = \boldsymbol{X}\boldsymbol{M}\boldsymbol{M}^+$.*

Proof. Note that $\boldsymbol{M}\boldsymbol{M}^+$ is the orthogonal projector onto $\text{range}(\boldsymbol{M})$. By definition, $\text{range}(\boldsymbol{M}) = \text{row}(\boldsymbol{X})$. Hence, for any $\boldsymbol{u} \in \text{row}(\boldsymbol{X})$, we have $\boldsymbol{M}\boldsymbol{M}^+\boldsymbol{u} = \boldsymbol{u}$. It follows that $\boldsymbol{M}\boldsymbol{M}^+\boldsymbol{X}^\top = \boldsymbol{X}^\top$. By transposing and noting that orthogonal projectors are symmetric, $\boldsymbol{X} = (\boldsymbol{M}\boldsymbol{M}^+\boldsymbol{X}^\top)^\top = \boldsymbol{X}\boldsymbol{M}\boldsymbol{M}^+$. □

B.2 Obtaining UWMs in Practice

We now provide more details about how the uncentered whitening matrices are obtained. We first begin with a discussion of past work.

The naive option to obtain an uncentered whitening matrix is to use the Cholesky decomposition of the second moments, as explained in Wang et al. (2025c). An issue is that we cannot compute it in the case of singular positive-semidefinite matrices. In their implementation[5], they address this issue by perturbing the matrix with random noise. Later, in Wang et al. (2025b), the authors note this issue and propose the use of an SVD-based whitening matrix, which supports data residing in a subspace (although their analysis does not seem to cover this case). Nevertheless, they show slight end-to-end accuracy gains coming from the SVD-based whitening.

As noted in Section 4.2, we use an eigendecomposition-based scheme, implemented in practice through the `eigh` routine in PyTorch (a routine that obtains the eigenvalues and eigenvectors of a Hermitian (symmetric in the case of $\mathbb{R}$) matrix, and is usually faster than calling the corresponding generic eigendecomposition routine). For notational convenience, we assume the eigenpairs are ordered so that the positive eigenvalues appear first. In implementation, this ordering is unnecessary because the construction and subsequent computations are invariant to a simultaneous permutation of the whitening coordinates.

Recall that, for a single group g, with $\boldsymbol{X} = \overline{I}(\mathbf{X})_g \in \mathbb{R}^{N \times I}$, we compute

$$[\boldsymbol{V}, \boldsymbol{\Lambda}] = \mathsf{eigh}(\frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X}),$$

and set

$$\boldsymbol{M} = \boldsymbol{V}\boldsymbol{\Lambda}^{+1/2}, \quad \boldsymbol{M}^+ = \boldsymbol{\Lambda}^{1/2}\boldsymbol{V}^\top,$$

where

$$\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top = \frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X}.$$

It is easy to show that it satisfies our definition. It follows that

$$\boldsymbol{\Lambda}^{+1/2}\boldsymbol{V}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{V} \boldsymbol{\Lambda}^{+1/2} = N\boldsymbol{\Lambda}^{+1/2}\boldsymbol{V}^\top \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\top \boldsymbol{V}\boldsymbol{\Lambda}^{+1/2} = N\overline{\boldsymbol{I}}_R,$$

[5] https://github.com/AIoT-MLSys-Lab/SVD-LLM

R being the rank of $\boldsymbol{X}$. Moreover, it is easy to see that $\text{row}(\boldsymbol{X}) = \text{range}(\boldsymbol{M})$, as required in our definition (see Appendix B.1). Additionally, the pseudoinverse of $\boldsymbol{\Lambda}$ (a diagonal matrix) can be obtained efficiently by inverting each nonzero element of the diagonal, while leaving zeros as is (in practice, we use a small numerical threshold). This avoids additional numerical approximation errors when forming $\boldsymbol{M}$. We do not perform any additional steps on the whitening matrices.

Practical considerations. We use the linear algebra functions (with CUDA tensors) provided in the PyTorch library. During early experimentation, we used the MAGMA solver (Heroux et al., 2024), which appeared to be more stable, but it was significantly slower than the default (cuSOLVER (NVIDIA, 2025)), which was ultimately used in our experiments. Sporadically, the `eigh` routine is unable to converge. For that reason, we add a fallback to the SVD-based approach described below. In none of our experiments did the fallback also fail (so we do not resort to noise addition in any instance, contrasting with Wang et al. (2025c)).

The alternative. An SVD-based UWM, as proposed in SVD-LLM V2 (Wang et al., 2025b), can be obtained in a similar way with the SVD of $\boldsymbol{X}^\top \boldsymbol{X}$. Although it is algebraically equivalent, it tends to be slower than our approach.

Accumulating the uncentered moments incrementally. The naive way of obtaining $\frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X}$ in practice is to use a single batch, and then cache activations by passing the batch through the network. However, for some networks with high-dimensional features and/or large batch sizes, the memory requirements might make the process expensive or impossible. A nice property (mentioned in Wang et al. (2025c), but not discussed extensively) is that we can use data batches to accumulate the second moments. Suppose we have M batches, then,

$$\frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X} = \frac{1}{N}\begin{bmatrix} \boldsymbol{X}_{1,:}^\top \\ \vdots \\ \boldsymbol{X}_{M,:}^\top \end{bmatrix} \left[\boldsymbol{X}_{1,:} \cdots \boldsymbol{X}_{M,:}\right] = \frac{1}{N}\left(\boldsymbol{X}_1^\top \boldsymbol{X}_1 + \cdots + \boldsymbol{X}_M^\top \boldsymbol{X}_M\right),$$

where N is the total outer dimension after $\overline{I}$ reshapes the activations (e.g., for convolutions, $N = BH_oW_o$). Here, the indexing denotes the selection of a sub-batch. We use this approach in our implementation.

C Rank Allocation Details

This section provides details about the rank allocation strategy discussed in Section 4.3.

Our objective is to select a list $\{P_l\}_{l=1}^{L}$ denoting the retained rank in each layer, with the objective of maximizing the global retained energy under some cost constraint. We can model this as a binary linear program in the form of a multiple-choice knapsack problem (for a review, see Szkaliczki (2025)), with decision variables $\{x_{l,t}\}_{l,t=1}^{L,U_l}$, where $x_{l,t} \in \{0,1\}$ denotes whether, at layer l, t is selected as the maximum retained rank, and U_l denotes an upper bound on the rank (in our case, the number of singular values). Furthermore, by establishing a function $C_l(P_l)$ that returns the computational cost (e.g., in FLOPs or number of parameters) of keeping up to rank P_l at layer l, we can ensure that the complexity of the solution is constrained to some user-set quantity. The binary program can be expressed as:

$$\begin{aligned}
\max_{\{x_{l,t}\}} \quad & \sum_{l=1}^{L}\sum_{t=1}^{U_l} E_l(t)\, x_{l,t} \\
\text{s.t.} \quad & \sum_{t=1}^{U_l} x_{l,t} = 1, && l = 1, \ldots, L, \\
& \sum_{l=1}^{L}\sum_{t=1}^{U_l} C_l(t)\, x_{l,t} \leq C_{\max}, \\
& x_{l,t} \in \{0,1\}, && \forall\, l, t.
\end{aligned}$$

Are there other options for the objective function? Recall that we use the total retained energy, which is normalized, as the objective function. Following the insights of Proposition 4.8 and Appendix D (although they are not tight for deeper networks), one might speculate that, with appropriate accounting of each layer's importance, we could achieve better results. During our initial experiments, we also evaluated unnormalized distortion as a metric in place of energy and explored layer-wise weightings based on statistics such as output variance and entropy. None of these alternatives improved performance over the method presented here; instead, they increased complexity and posed additional computational overhead and implementation challenges.

Measuring the cost. Our allocator makes use of a cost function. We refer the reader to Appendix A for the discussion on the cost of regular and factorized layers, both in terms of FLOPs and in terms of the number of parameters. It is clear that not all P_l selections result in fewer parameters or FLOPs. Given P_l and a criterion, we decompose the layer only when we achieve gains in terms of that criterion (e.g., a reduction in FLOPs or parameters); see Algorithm 1. To reflect this in the cost function, we define it as a piecewise linear function, where the first region is monotonically increasing and represents the region where decomposing the layer results in compression, and the second remains constant, denoting that the action of "keeping rank P_l" will have the cost of the original layer. Summarizing, we define

$$C_l(P_l) = \min\{C_l^{\text{orig}}, C_l^{\text{decomp}}(P_l)\}.$$

We emphasize that this step is important so as to inform the solver that we will not factorize a layer if it is not beneficial.

C.1 A Scalable Lagrangian Relaxation

The aforementioned program can be solved with publicly available libraries (e.g., PuLP if using Python (Dunning et al., 2011)). The issue is that solving it exactly is known to be NP-hard (see Szkaliczki (2025)). To overcome this, we employ a Lagrangian relaxation. Pseudocode defining our implementation is given in Algorithm 2. It accepts a hyperparameter I denoting the number of iterations to use. For a more general discussion, see Kellerer et al. (2004, Chapter 11). In particular, for a fixed multiplier λ, our per-layer update solves the relaxed subproblem in Kellerer et al. (2004, Equation (11.14)): each layer independently selects the rank maximizing $E_l(t) - \lambda C_l(t)$. We then search over λ by bisection, using the monotonicity of the induced total cost, and return the feasible allocation obtained at the final multiplier. As usual for such a relaxation-based heuristic, the returned primal allocation need not be globally optimal.

Complexity. Each of the I outer iterations scans all the candidates once, giving time $O(I \sum_l U_l)$. The initial doubling to find $\lambda_{\max}$ adds an additional $O(D \sum_l U_l)$ factor. Thus, the total time complexity is $O((I + D + 1) \sum_l U_l)$ (in the case of a nontrivial solution).

Quality of the solutions. The algorithm yields an approximate solution, but in every experiment we carried out, it produced good results and saturated (approximately) the target budget. It is, nevertheless, interesting to visualize how the algorithm behaves w.r.t. the number of iterations. Figure 4 compares the number of iterations with two quantities of interest: model accuracy and the actual FLOPs used by the model (when using it with the objective of keeping 60% of the FLOPs).

Undershooting. We describe a specific situation in which our allocator can undershoot the budget. The allocator does not explicitly maximize budget utilization; it maximizes retained energy subject to an upper bound cost constraint. Consequently, the returned solution may use strictly less than the target budget. A simple case occurs when, for each layer, the retained energy curve saturates at some effective rank. Since the algorithm breaks ties in favor of the smallest-cost maximizer, selection will not choose ranks larger than these minimal saturating ranks. If their total cost is already under the budget, the algorithm will not saturate the budget. Although we did not observe this behavior in our experiments, it can theoretically occur for networks with very sharp spectra, for example, as a result of low-rank regularization (Ghosh et al., 2026).

Algorithm 2 Lagrangian relaxation for our rank allocator

Require: L layers, each with U_l choices;
energy function $E_l(\cdot)$ and costs $C_l(\cdot)$; total budget $C_{\max}$;
max iterations I (e.g., $I = 300$ in our experiments);
Assumptions (in our problem instance): (1) the problem is feasible (at least one selection with total cost $\leq C_{\max}$ exists); (2) all costs and energies are nonnegative; (3) within each layer l, costs and energies are nondecreasing.
Ensure: Selection t_l for each layer l, one index $t_l \in \{1, \ldots, U_l\}$.

```
λ_min ← 0, λ_max ← 1                                   ▷ initialize dual bounds
for each layer l = 1, ..., L do
    t_l^(0) ← min(arg max_t E_l(t))                    ▷ same as λ = 0; break ties with smallest cost
end for
C^(0) ← Σ_l C_l(t_l^(0))
if C^(0) ≤ C_max then
    return {t_l^(0)}                                   ▷ the unconstrained max-energy selection is already feasible
end if
repeat                                                 ▷ grow λ_max until feasibility at λ_max
    for each layer l = 1, ..., L do
        t_l ← min(arg max_t (E_l(t) − λ_max C_l(t)))   ▷ break ties with smallest cost
    end for
    C ← Σ_l C_l(t_l)
    if C > C_max then
        λ_max ← 2λ_max
    end if
until C ≤ C_max                                        ▷ now λ_max yields a feasible selection
for k = 1 to I do                                      ▷ bisection to shrink the feasible/infeasible bracket
    λ ← (λ_min + λ_max)/2
    for each layer l = 1, ..., L do
        t_l ← min(arg max_t (E_l(t) − λ C_l(t)))       ▷ break ties with smallest cost
    end for
    C ← Σ_l C_l(t_l)
    if C > C_max then
        λ_min ← λ
    else
        λ_max ← λ                                      ▷ keep smallest feasible λ
    end if
end for                                                ▷ bracket invariant: C(λ_min) > C_max, C(λ_max) ≤ C_max
for each layer l = 1, ..., L do
    t_l ← min(arg max_t (E_l(t) − λ_max C_l(t)))
end for
C ← Σ_l C_l(t_l)
return {t_l}                                           ▷ final feasible selection, good in practice
```

Implementation details. We implement the algorithm in Python, as even for larger models, it has a negligible runtime (less than 0.2 seconds in all our experiments, see Appendix G). Moreover, although the energy and cost functions are represented in functional form in the algorithm description, in practice, they are precomputed as vectors for each layer.

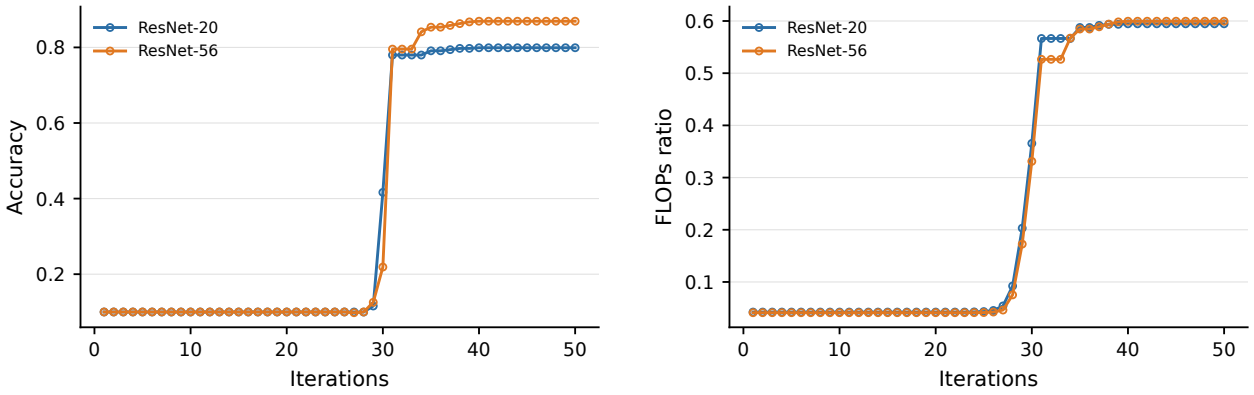

Figure 4: The effect of the number of iterations on CIFAR-10 models. We fix a compression target of 60% of the original FLOPs. The horizontal axis denotes the number of iterations. The vertical axes denote the accuracy achieved (left) and the number of used FLOPs (right). The results stabilize with a relatively small number of iterations.

D Bounding the Output Distortion of a Model

In this section, we provide a more thorough analysis of the claim in Proposition 4.8. We provide a more complete description and its proof. The result applies to sequential (possibly nonlinear) networks with $(\overline{O}, \overline{I}, \overline{P})$-expressible layers (possibly with biases).

Define a sequential L-layer network

$$\mathbf{X}^0 = \mathbf{X}, \quad \mathbf{X}^l = a_l\left(f_l(\mathbf{X}^{l-1}; \mathbf{W}^l) + \boldsymbol{b}^l\right) = a_l\left(\overline{O}_l(\overline{I}_l(\mathbf{X}^{l-1})\overline{P}_l(\mathbf{W}^l)) + \boldsymbol{b}^l\right),$$

where a_l is an element-wise activation function with Lipschitz constant A_l. Moreover, let B_l be the Lipschitz constant of $\overline{I}_l$ (note that $\overline{I}_l$ is linear and thus Lipschitz-continuous). We restrict our analysis to the case of ungrouped layers. When approximating $\mathbf{W}^l$ as a low-rank version, we denote the resulting parameter by $\widehat{\mathbf{W}}^l$, and the corresponding cumulatively distorted intermediate activations as $\widehat{\mathbf{X}}^l$.

For each layer l, fix the retained rank and let

$$\sqrt{\ell_l^{\text{activ}}} = \frac{1}{\sqrt{B}}\left\| f_l(\mathbf{X}^{l-1}; \mathbf{W}^l) - f_l(\mathbf{X}^{l-1}; \widehat{\mathbf{W}}^l)\right\|_F,$$

which reads as the (non-cumulative) error introduced by perturbing $\mathbf{W}^l \to \widehat{\mathbf{W}}^l$. Then, the total output distortion

$$d^{(1:L)} = \frac{1}{B}\|\mathbf{X}^L - \widehat{\mathbf{X}}^L\|_F^2$$

satisfies the bound

$$\sqrt{d^{(1:L)}} \leq \sum_{l=1}^{L}\left[A_l\sqrt{\ell_l^{\text{activ}}}\prod_{i=l+1}^{L}\|\overline{P}_i(\widehat{\mathbf{W}}^i)\|_2 A_i B_i\right].$$

We follow the convention that the empty product is equal to 1.

Proof. For brevity, we set $\boldsymbol{W}^l \equiv \overline{P}_l(\mathbf{W}^l)$ (recall that we assume ungrouped layers).

Note that $\|\overline{O}_l(\cdot)\|_F = \|\cdot\|_F$, as $\overline{O}_l(\cdot)$ is a composition of permutations and reshapes. It is also true that $a_l \circ \overline{O}_l = \overline{O}_l \circ a_l$, as a_l is element-wise.

Then:

$$
\begin{aligned}
\sqrt{Bd^{(1:L)}} &= \|\mathbf{X}^L - \widehat{\mathbf{X}}^L\|_F \\
&= \|a_L(f_L(\mathbf{X}^{L-1}; \mathbf{W}^L) + \boldsymbol{b}^L) - a_L(f_L(\widehat{\mathbf{X}}^{L-1}; \widehat{\mathbf{W}}^L) + \boldsymbol{b}^L)\|_F \\
&= \|a_L(\overline{O}_L(\overline{I}_L(\mathbf{X}^{L-1})\boldsymbol{W}^L) + \boldsymbol{b}^L) - a_L(\overline{O}_L(\overline{I}_L(\widehat{\mathbf{X}}^{L-1})\widehat{\boldsymbol{W}}^L) + \boldsymbol{b}^L)\|_F \\
&\leq A_L\|\overline{O}_L(\overline{I}_L(\mathbf{X}^{L-1})\boldsymbol{W}^L) + \boldsymbol{b}^L - \overline{O}_L(\overline{I}_L(\widehat{\mathbf{X}}^{L-1})\widehat{\boldsymbol{W}}^L) - \boldsymbol{b}^L\|_F \\
&= A_L\|\overline{I}_L(\mathbf{X}^{L-1})\boldsymbol{W}^L - \overline{I}_L(\widehat{\mathbf{X}}^{L-1})\widehat{\boldsymbol{W}}^L\|_F \\
&= A_L\|\overline{I}_L(\mathbf{X}^{L-1})\boldsymbol{W}^L - \overline{I}_L(\mathbf{X}^{L-1})\widehat{\boldsymbol{W}}^L + \overline{I}_L(\mathbf{X}^{L-1})\widehat{\boldsymbol{W}}^L - \overline{I}_L(\widehat{\mathbf{X}}^{L-1})\widehat{\boldsymbol{W}}^L\|_F \\
&\leq A_L\|\overline{I}_L(\mathbf{X}^{L-1})\boldsymbol{W}^L - \overline{I}_L(\mathbf{X}^{L-1})\widehat{\boldsymbol{W}}^L\|_F + A_L\|\overline{I}_L(\mathbf{X}^{L-1})\widehat{\boldsymbol{W}}^L - \overline{I}_L(\widehat{\mathbf{X}}^{L-1})\widehat{\boldsymbol{W}}^L\|_F \\
&= A_L\sqrt{B\ell_L^{\text{activ}}} + A_L\|\overline{I}_L(\mathbf{X}^{L-1})\widehat{\boldsymbol{W}}^L - \overline{I}_L(\widehat{\mathbf{X}}^{L-1})\widehat{\boldsymbol{W}}^L\|_F \\
&\leq A_L\sqrt{B\ell_L^{\text{activ}}} + \|\widehat{\boldsymbol{W}}^L\|_2 A_L B_L\|\mathbf{X}^{L-1} - \widehat{\mathbf{X}}^{L-1}\|_F \\
&= A_L\sqrt{B\ell_L^{\text{activ}}} + A_L B_L\|\widehat{\boldsymbol{W}}^L\|_2\sqrt{Bd^{(1:L-1)}}.
\end{aligned}
$$

By unrolling the recursion and dividing by $\sqrt{B}$, one arrives at the claimed statement.

□

Some special cases. There are some interesting instances where the bound reduces to simpler expressions. For example, for fully connected layers, $B_l = 1$. Moreover, for 1-Lipschitz activations, such as ReLU (Nair & Hinton, 2010), $A_l = 1$. For instance, on MLPs with ReLU activations, the bound reduces to

$$
\sqrt{d^{(1:L)}} \leq \sum_{l=1}^{L}\left[\sqrt{\ell_l^{\text{activ}}}\prod_{i=l+1}^{L}\|\widehat{\boldsymbol{W}}^i\|_2\right].
$$

Looseness of the bound. In deep networks, as noted in the main text, this bound is informative rather than practical, as it tends to become looser with depth; the main reason is that the bound involves products of spectral norms. There are some works, for instance Cisse et al. (2017), that aim to constrain the operator norms of the layers of neural networks. Such constraints would directly tighten our result, but they are not enforced in standard architectures.

E Experimental Details

This section provides additional experimental details. All experiments were carried out using the PyTorch library. We make our code public at `https://anonymous.4open.science/r/BALF-2AFC`. To ensure reproducibility, we include (1) the library code that implements our methods, as well as the baselines, (2) the Python scripts used to carry out the experiments, (3) the Bash scripts used to call the Python scripts with the specific hyperparameters used, and (4) the dependencies and versions used.

E.1 CIFAR-10 Experiments

Models. Since there are no widely adopted pre-trained CIFAR-10 ResNet model weights, we pre-train the ResNet-20 and ResNet-56 models ourselves, following standard practice. Our CIFAR-10 ResNet implementation is based on the fb.resnet.torch codebase, with the shortcut option set to B: the residual branch uses an identity mapping when the input and output shapes match, and otherwise applies a 1×1 convolution with the appropriate stride followed by batch normalization to match shapes. We use the default PyTorch parameter initialization for all layers.

We use SGD with momentum 0.9, weight decay 10^{-4}, and a batch size of 128. The learning rate is initialized to 0.1 and reduced by a factor of 0.1 at epochs 100 and 150. We save the final model checkpoint at epoch

200. For data augmentation, we apply 4-pixel zero-padding followed by a random crop to 32×32 pixels, a random horizontal flip with a probability of 0.5, and per-channel normalization. Under these settings, the final test accuracies are 91.93% for ResNet-20 and 93.18% for ResNet-56.

Compression. In order to obtain the k-calibration dataset, we pick k random images uniformly from the training dataset. For all our main experiments, we use 1024 images.

All factorizable layers are considered for compression. For the CIFAR-10 ResNet models, this includes all the convolutional layers, including shortcut layers, and the final fully connected classifier.

Dataset processing. We do not use preprocessing in our calibration dataset and test dataset, other than normalization, so as to mimic inference behavior.

E.2 ImageNet-1K Experiments

Models. For all the CNNs, we use the implementation provided by the `torchvision` library (TorchVision maintainers and contributors, 2016). We also use their `V1` checkpoints for all the models. For the experiments on vision transformers, we use the implementation and weights provided in the `timm` library (Wightman, 2019).

Compression. In order to obtain the k-calibration dataset, we pick k random images uniformly from the training dataset. For all our main experiments, we use 8192 images.

All factorizable layers are considered for compression. For the CNNs, this includes all their convolutional layers (including shortcuts) and the linear layers present in the classifier. For the vision transformers, this includes the initial convolutional layer, as well as the linear layers inside attention blocks and MLPs. For the attention blocks, we factorize the merged Q-K-V matrix (implemented in `timm` as a single linear layer).

Dataset processing. Following standard inference practices, we first resize images to 248×248 for vision transformers (using bicubic interpolation) and to 256×256 (using bilinear interpolation) for the remaining models, and then center-crop them to 224×224. We then normalize them following the standard procedure. In particular, we use the default ImageNet-1K mean and standard deviations for all models except for the ViT-B/16, for which we follow the conventions of `timm` and use 0.5 as the mean and standard deviation for all channels. We do this for both the calibration dataset and the validation dataset.

E.3 General Details

For all our main experiments, we use 300 iterations for our rank allocator (see Appendix C.1). For all the compression sweeps in our experiments section, we conduct a full sweep over the compression targets. For the experiments with our allocator, we select compression ratios (w.r.t. parameter and FLOP counts) between 0.1 and 1.0 in steps of 0.05; additionally, we add 0.975. For the other methods (energy-based methods), we manually select an appropriate set of hyperparameters for each model so that they cover a suitable spectrum of compression ratios.

We follow standard practice when reporting compression results. For FLOP counts, we account for convolutional and fully connected layers, but not other secondary layers, like normalization layers. We note that, for ViT models, attention FLOPs are also counted in the reported results but are not taken into account by the allocator for simplicity. For parameter count results, we include all network parameters, including those that are not compressed (which are typically negligible; similar to the FLOP counts, these are not taken into account by the allocator). Finally, unless otherwise noted, we do not perform any layer fusion (e.g., BN+convolution fusion).

For some comparison results in the experimental section, we used WebPlotDigitizer[6] to extract numeric values from figures when the values were not reported numerically in tables. We took care to perform the extraction as accurately as possible.

[6] https://automeris.io/wpd

E.4 Computational Resources

Earlier runs of the experiments were executed on a laptop with an RTX 2070 and a node with an RTX 4090. The experiments presented in the final version were each run on a single A100 40GB GPU. Preliminary experimentation that did not make it into the paper was conducted on a variety of hardware configurations, including an A100 40GB and other RTX GPUs.

F Additional Experimental Results

This section presents some additional experimental results that aim to answer secondary questions.

F.1 How Impactful is the Calibration Dataset Size?

Although we used 8192 calibration images for ImageNet-1K and 1024 for CIFAR-10, we noticed that moderate calibration dataset sizes worked well. Here, we carry out a sweep over different calibration dataset sizes and compression ratios with parameter compression targets. We use a ResNet-20 and ResNet-56 on CIFAR-10, and ResNet-18 and ResNet-50 on ImageNet-1K. Figure 5 shows the different results. As shown, CIFAR-10 is not particularly sensitive to the size of the calibration dataset. On the other hand, the need for larger calibration sets is more apparent on ImageNet-1K. This need increases for larger models (in our case, the ResNet-50), and is exacerbated for higher compression ratios. However, beyond 1024–2048 images, more images yield only marginally better results.

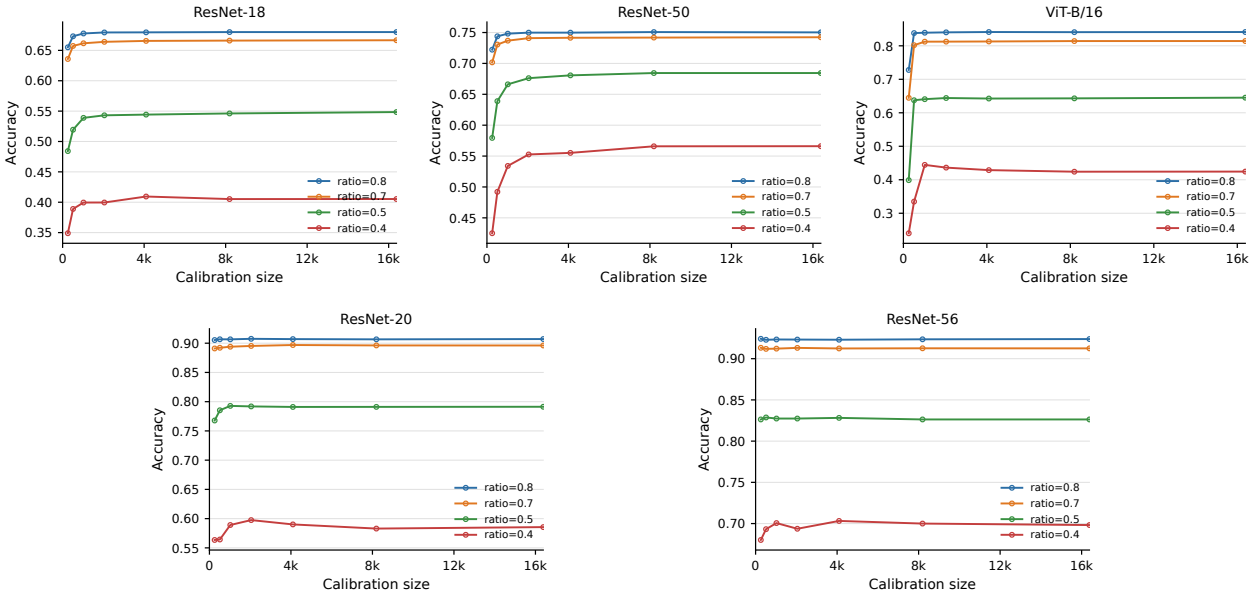

Figure 5: Calibration size vs. performance across models. Hyperparameters are fixed to those of the main experiments, with calibration sizes swept over $256, 512, 1024, 2048, 4096, 8192, 16384$. Ratios indicate the target parameter ratios passed to the allocator.

F.2 Robustness under Distribution Shift

Our method relies on the computation of (uncentered) whitening matrices (over the training dataset in our main experiments) to estimate the optimal low-rank projections. We rely on the i.i.d. assumption to expect transfers to unseen data. As shown over the training–test pairs on CIFAR-10 and ImageNet-1K, we obtain good results. In general, the i.i.d. assumption is common (although seldom discussed) in methods that use calibration data.

Here, we evaluate BALF under calibration–evaluation mismatch. The base models are unchanged and are trained on the original training datasets. We then compute the calibration statistics using shifted data, but evaluate the resulting compressed models on the standard clean evaluation sets. Thus, these experiments test whether BALF remains effective when the calibration data used for compression differs from the data distribution used at evaluation time.

For CIFAR-10, we use ResNet-20 and draw calibration data from CIFAR-10-C (Hendrycks & Dietterich, 2019), while evaluating on the clean CIFAR-10 test set. For ImageNet-1K, we use ResNet-18 and draw calibration data from ImageNet-C (Hendrycks & Dietterich, 2019) or ImageNet-V2 (Recht et al., 2021), while evaluating on the standard ImageNet-1K validation set. Results for CIFAR-10 appear in Figure 7, results for ImageNet-C appear in Figures 8 and 9, and results for ImageNet-V2 appear in Figure 10.

For corruption-based datasets, the results vary substantially depending on the corruption type and severity. While some have minor effects on the effectiveness of BALF, others substantially degrade performance. However, performance on the ImageNet-V2 experiment suggests that BALF is robust under natural distribution shifts.

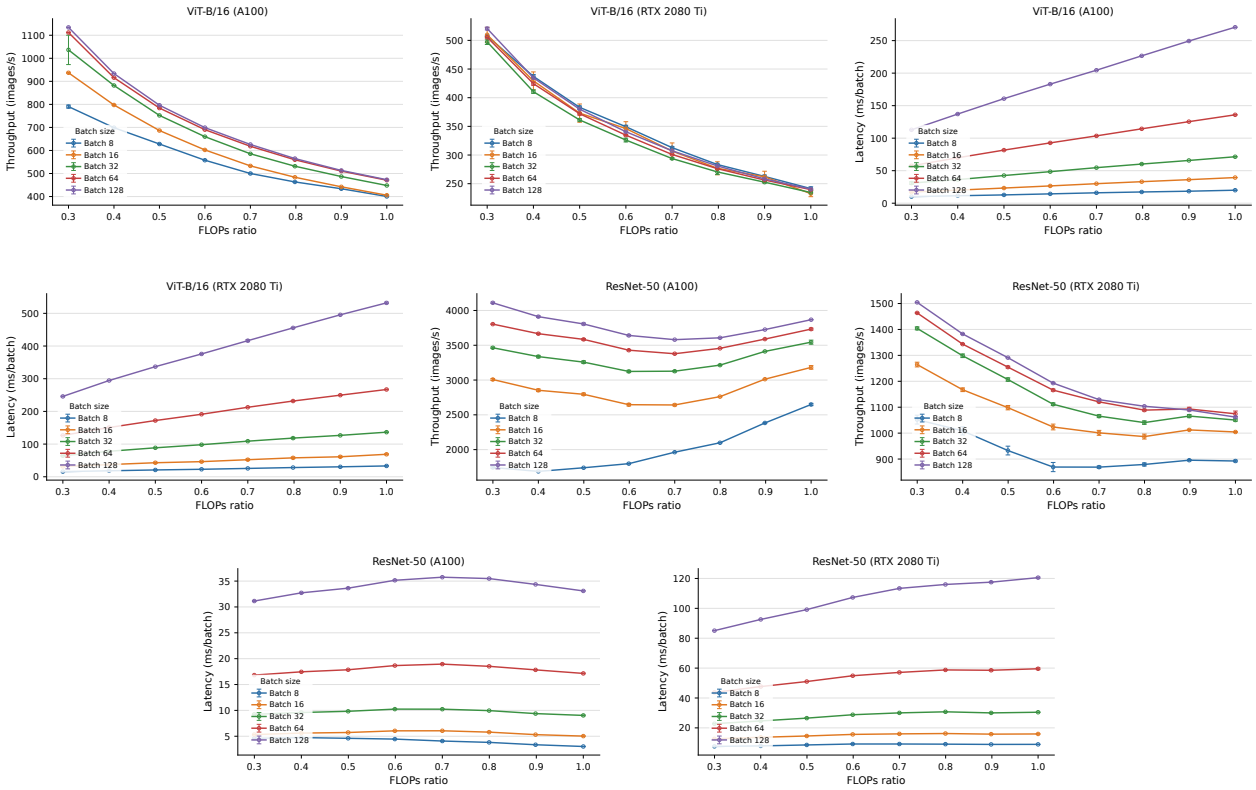

Figure 6: Throughput and latency as a function of FLOP ratio for ViT-B/16 and ResNet-50 on NVIDIA A100 and RTX 2080 Ti GPUs.

F.3 Speedup of Low-Rank Layers in Practice

This section discusses the relationship between our compression results (in terms of FLOPs) and the end-to-end speedup obtained in practice. We use basic PyTorch implementations of low-rank operators. In particular, we substitute each original layer with two sequential layers, as described in Appendix A, without any specialized implementation.

We measure the throughput (and latency for completeness) of two ImageNet-1K models (ResNet-50 and ViT-B/16) on an RTX 2080 Ti and an A100 GPU at different compression ratios. Every batch normalization

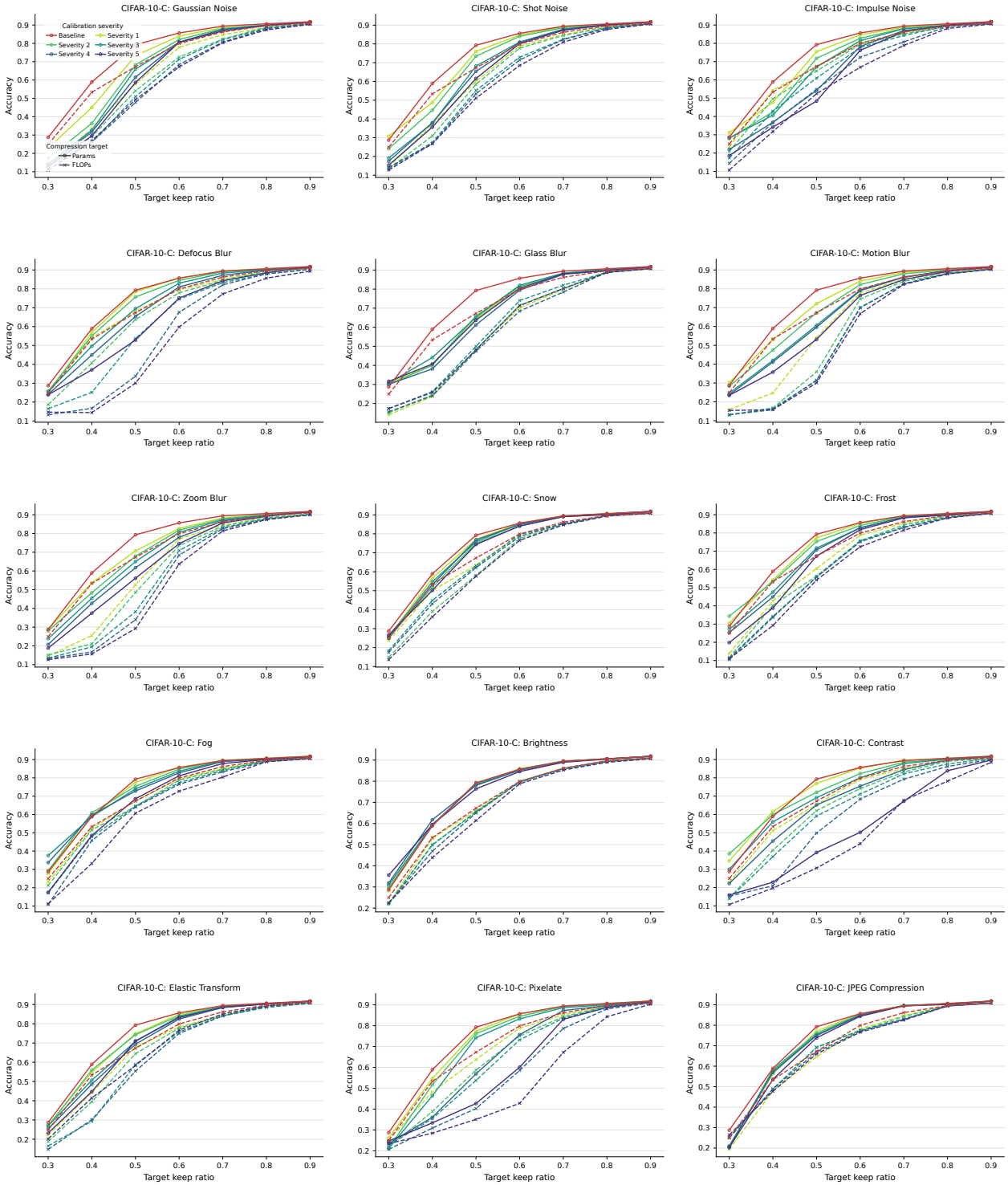

Figure 7: Compression–accuracy curves on ResNet-20 when calibration data are drawn from CIFAR-10-C and evaluation is performed on the clean CIFAR-10 test set. Each panel corresponds to one CIFAR-10-C corruption type, with curves shown across target keep ratios and calibration severities.

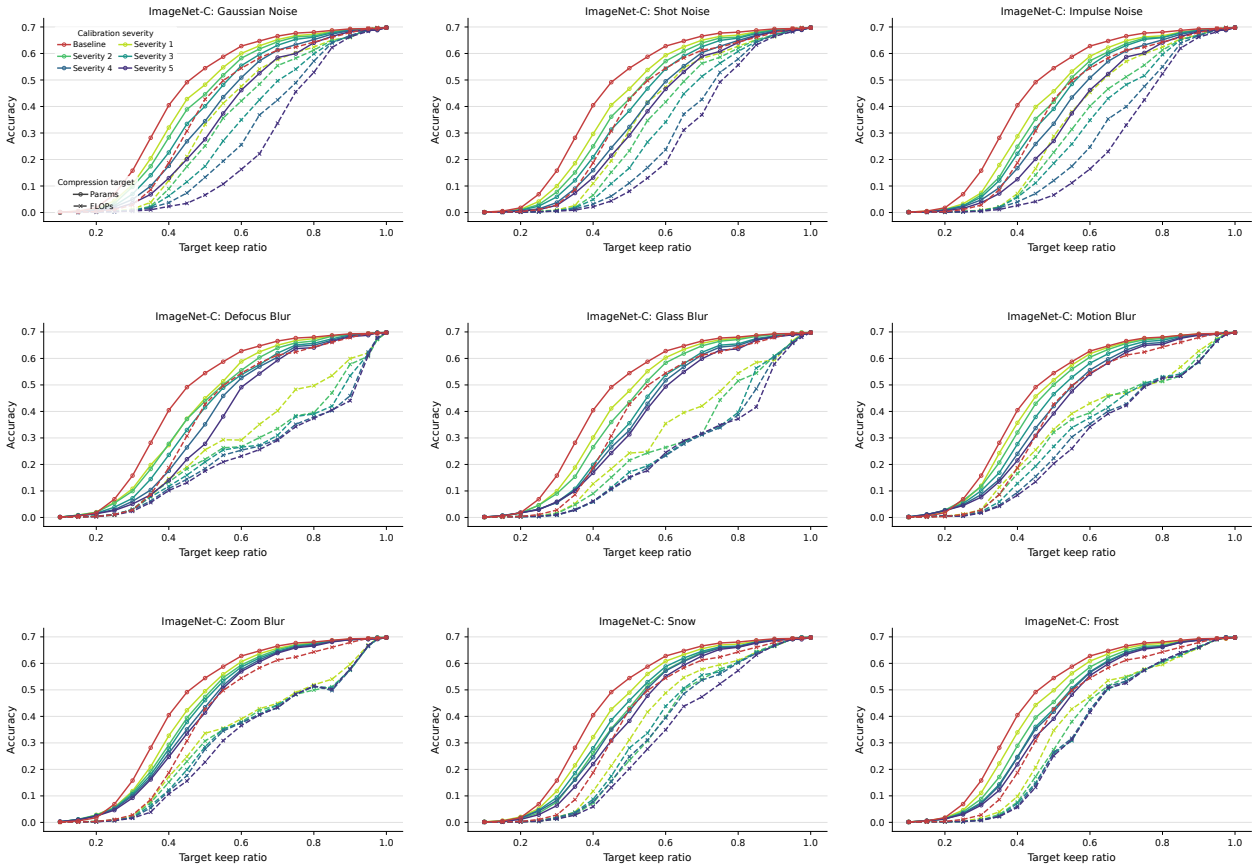

Figure 8: Compression–accuracy curves on ResNet-18 when calibration data are drawn from ImageNet-C and evaluation is performed on the standard ImageNet-1K validation set. Each panel corresponds to one ImageNet-C corruption type from the noise, blur, and selected weather categories.

(BN) (Ioffe & Szegedy, 2015) layer is first removed from the ResNet-50 model so as to mimic the usual inference scenario (where BN layers are folded). We note that our method does not interfere with BN folding in any way.

We found that compression results translate well to actual speedups (see Figure 6) in terms of throughput only in some circumstances. While matrix-multiplication-dominant models such as ViT obtain good speedups, it is important to note that convolutional layers fare worse than linear layers in this regard, and that the theoretical gains are not fully exploited. In particular, on the A100 GPU, no consistent speedup is obtained for ResNet-50.

We expect that specialized operator implementations, including those found in software for edge devices, might be able to fully exploit the compression in terms of speedup. In fact, Sui et al. (2024), for instance, were able to obtain near-perfect practical speedups from FLOP reductions on ASICs and FPGAs on ResNet-50 with low-rank layers.

We note that this is, in general, a persistent problem with compression methods (see, e.g., Narshana et al. (2023), where, without specialized implementations, the actual speedups are significantly lower than the FLOPs reductions). We leave the exploration of more efficient implementations for future work.

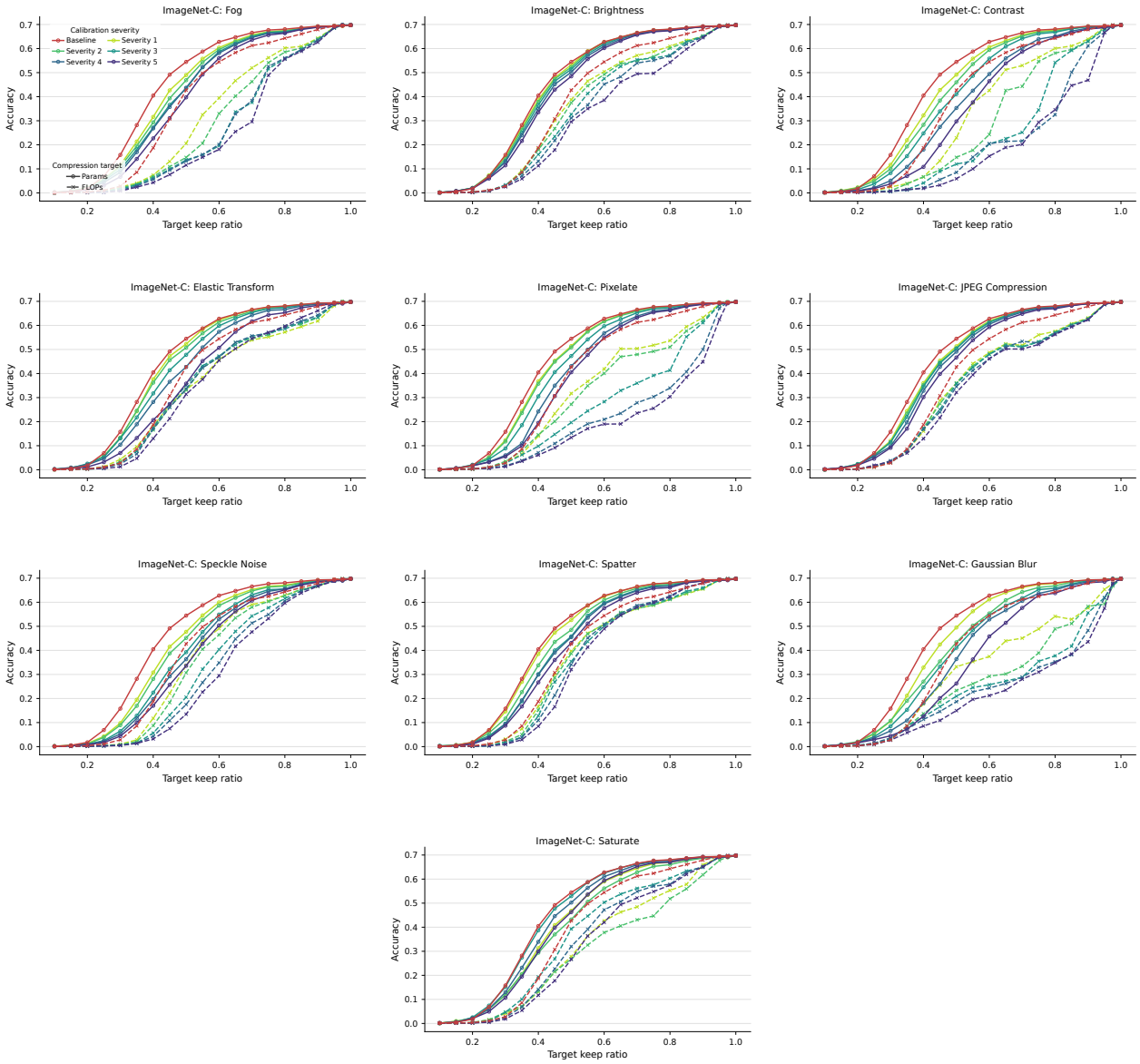

Figure 9: Compression–accuracy curves on ResNet-18 when calibration data are drawn from ImageNet-C and evaluation is performed on the standard ImageNet-1K validation set. Each panel corresponds to one ImageNet-C corruption type from the remaining weather, digital, and extra corruption categories.

G Runtime and Memory Footprint of BALF

Our implementation of BALF is conceptually similar to Algorithm 1, but there are some implementation details that are important to discuss. We first provide a high-level overview of the implementation details:

1. First, given a calibration dataset, we gather the activation moments. The model runs on the GPU, but the activation moments are offloaded to CPU memory to reduce GPU memory usage and allow compression on lower-end GPUs. Once activation moments are gathered, they can be cached (on disk) for subsequent compression configurations.

2. Then, whitening matrices and factors are computed on the GPU. These are cached on disk to maintain low memory usage and to avoid recomputation in future runs.

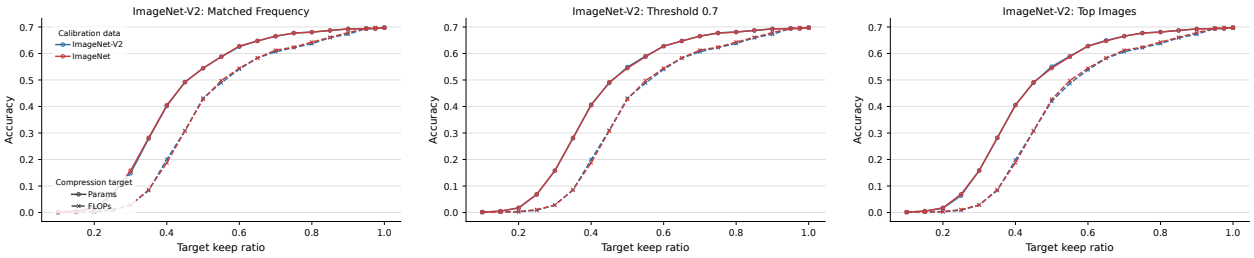

Figure 10: Compression–accuracy curves on ResNet-18 when calibration data are drawn from ImageNet-V2 and evaluation is performed on the standard ImageNet-1K validation set. Each panel corresponds to one ImageNet-V2 variant.

3. Our rank allocator uses the singular values (computed in the preceding step) to decide how much to compress each layer.

4. Then, we iterate through the model and substitute the layers with their truncated versions.

Table 3: Runtime breakdown and memory usage for different models. All timings are in seconds. Act. denotes activation-moment collection, Fact.+Whit. denotes factorization and whitening, Replace denotes layer replacement, Misc denotes miscellaneous overhead, and Peak Mem. denotes peak CUDA memory usage during the run in GiB. We report mean and standard deviation across 3 runs.

Model	Act.	Fact.+Whit.	Solver	Replace	Misc	Total	Peak Mem.
ResNet-18	$78.83_{\pm 0.41}$	$5.74_{\pm 0.11}$	$0.03_{\pm 0.01}$	$0.72_{\pm 0.03}$	$0.02_{\pm 0.00}$	$85.35_{\pm 0.26}$	$1.16_{\pm 0.00}$
ResNet-50	$97.65_{\pm 0.41}$	$5.32_{\pm 0.28}$	$0.06_{\pm 0.00}$	$0.96_{\pm 0.01}$	$0.00_{\pm 0.00}$	$103.98_{\pm 0.48}$	$1.34_{\pm 0.00}$
MobileNetV2	$41.54_{\pm 0.77}$	$0.89_{\pm 0.09}$	$0.05_{\pm 0.00}$	$0.14_{\pm 0.00}$	$0.00_{\pm 0.00}$	$42.61_{\pm 0.68}$	$2.23_{\pm 0.00}$
ResNeXt-50 (32×4d)	$56.97_{\pm 0.93}$	$3.91_{\pm 0.06}$	$0.06_{\pm 0.01}$	$0.41_{\pm 0.01}$	$0.00_{\pm 0.00}$	$61.35_{\pm 0.98}$	$2.29_{\pm 0.00}$
ResNeXt-101 (32×8d)	$181.22_{\pm 1.85}$	$19.85_{\pm 0.44}$	$0.13_{\pm 0.00}$	$1.58_{\pm 0.08}$	$0.01_{\pm 0.00}$	$202.79_{\pm 1.92}$	$4.45_{\pm 0.00}$
ViT-B/16	$112.14_{\pm 0.47}$	$10.59_{\pm 0.19}$	$0.11_{\pm 0.03}$	$1.33_{\pm 0.12}$	$0.00_{\pm 0.00}$	$124.17_{\pm 0.57}$	$0.83_{\pm 0.00}$
DeiT-B/16	$122.44_{\pm 6.79}$	$9.87_{\pm 0.24}$	$0.06_{\pm 0.00}$	$1.20_{\pm 0.06}$	$0.00_{\pm 0.00}$	$133.57_{\pm 7.07}$	$0.83_{\pm 0.00}$

For a thorough examination of the implementation, we refer the reader to our code[7]. Table 3 presents the runtime breakdown and peak memory usage of our method on different models on the ImageNet-1K dataset, evaluated on an RTX 2080 Ti GPU. We use 8192 calibration samples (as in our main experiments), divided into batches of 64 samples. All other parameters match those used in the main experiments. We note that the peak memory usage typically occurs during the computation of activation moments, and that using smaller batch sizes can further reduce it.

The only hyperparameters that are expected to affect the runtime of our method are the number of calibration samples and the number of solver iterations. For the first, it is only expected to affect (linearly) the activation-gathering phase (Act. in Table 3).

Second, the number of solver iterations affects (linearly) the time it takes to produce a solution (Solver in Table 3). Its runtime is negligible (less than 0.15 seconds in the runs of this section, using 300 iterations).

Finally, we reiterate that the most expensive steps (activation moment gathering and factorization/whitening) are cached, allowing one to obtain different models with different compression configurations in a matter of seconds on average, even on lower-end hardware, after the first run. We also note that our implementation was not particularly optimized (as compression time is not a bottleneck in our evaluations), and that further runtime gains are probably attainable.

[7] https://anonymous.4open.science/r/BALF-2AFC

H A Post-Hoc Interpretation via Shumaylov et al. (2025)

Recent work by Shumaylov et al. (2025) provides interesting insights into the factorization of neural networks from an information-geometric point of view. BALF has an interesting interpretation through the lens of their framework.

Let $\widehat{\mathbf{W}}$ denote the low-rank approximation of a parameter tensor. They posit that general activation-aware factorization approaches solve an optimization problem equivalent to minimizing the squared output mismatch under an i.i.d. Gaussian noise model with identity covariance. Translating that to our framework, we obtain

$$\begin{aligned}
&\arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}}} \sum_{l=1}^{L} \left\| \overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\mathbf{W}^l) - \overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\widehat{\mathbf{W}}^l) \right\|_F^2 \\
&= \arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}}} \sum_{l=1}^{L} \sum_{g=1}^{G_l} \left\| \overline{I}_l(\mathbf{X}^{l-1})_{g,:,:} \overline{P}_l(\mathbf{W}^l)_{g,:,:} - \overline{I}_l(\mathbf{X}^{l-1})_{g,:,:} \overline{P}_l(\widehat{\mathbf{W}}^l)_{g,:,:} \right\|_F^2 \\
&= \arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}}} \sum_{l=1}^{L} \sum_{g=1}^{G_l} \sum_{n=1}^{N_l} \left\| \overline{I}_l(\mathbf{X}^{l-1})_{g,n,:} \overline{P}_l(\mathbf{W}^l)_{g,:,:} - \overline{I}_l(\mathbf{X}^{l-1})_{g,n,:} \overline{P}_l(\widehat{\mathbf{W}}^l)_{g,:,:} \right\|_2^2 \\
&= \arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}}} D_{\mathrm{KL}}\big(p_{\mathbf{W}} \,\|\, p_{\widehat{\mathbf{W}}}\big),
\end{aligned}$$

with

$$p_{\mathbf{W}}(\cdot) = \prod_{l=1}^{L} \prod_{g=1}^{G_l} \prod_{n=1}^{N_l} \mathcal{N}\Big(\cdot \,\Big|\, \overline{I}_l(\mathbf{X}^{l-1})_{g,n} \overline{P}_l(\mathbf{W}^l)_g, \boldsymbol{I}\Big),$$

and

$$p_{\widehat{\mathbf{W}}}(\cdot) = \prod_{l=1}^{L} \prod_{g=1}^{G_l} \prod_{n=1}^{N_l} \mathcal{N}\Big(\cdot \,\Big|\, \overline{I}_l(\mathbf{X}^{l-1})_{g,n} \overline{P}_l(\widehat{\mathbf{W}}^l)_g, \boldsymbol{I}\Big).$$

In the preceding expressions, the n index corresponds to the outer dimension (e.g., with $BH_oW_o = N$ for convolutional layers), g to the groups, and l to the layers. The optimization problem generally assumes ranks are fixed ($\widehat{\mathcal{M}}$ denotes a model with layers truncated to fixed ranks, assumed to be given in advance), and it is only concerned with finding the best low-rank projections.

On the other hand, the problem BALF solves can be formulated as

$$\begin{aligned}
&\arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}} : C(\widehat{\mathcal{M}}) \le C_{\max}} \sum_{l=1}^{L} \frac{\left\| \overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\mathbf{W}^l) - \overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\widehat{\mathbf{W}}^l) \right\|_F^2}{\left\| \overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\mathbf{W}^l) \right\|_F^2} \\
&= \arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}} : C(\widehat{\mathcal{M}}) \le C_{\max}} \sum_{l=1}^{L} \sum_{g=1}^{G_l} \frac{\left\| \overline{I}_l(\mathbf{X}^{l-1})_{g,:,:} \overline{P}_l(\mathbf{W}^l)_{g,:,:} - \overline{I}_l(\mathbf{X}^{l-1})_{g,:,:} \overline{P}_l(\widehat{\mathbf{W}}^l)_{g,:,:} \right\|_F^2}{\left\| \overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\mathbf{W}^l) \right\|_F^2} \\
&= \arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}} : C(\widehat{\mathcal{M}}) \le C_{\max}} \sum_{l=1}^{L} \sum_{g=1}^{G_l} \sum_{n=1}^{N_l} \frac{\left\| \overline{I}_l(\mathbf{X}^{l-1})_{g,n,:} \overline{P}_l(\mathbf{W}^l)_{g,:,:} - \overline{I}_l(\mathbf{X}^{l-1})_{g,n,:} \overline{P}_l(\widehat{\mathbf{W}}^l)_{g,:,:} \right\|_2^2}{\left\| \overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\mathbf{W}^l) \right\|_F^2} \\
&\qquad = \arg \min_{\{\widehat{\mathbf{W}}^l\} \in \widehat{\mathcal{M}} : C(\widehat{\mathcal{M}}) \le C_{\max}} D_{\mathrm{KL}}\big(p_{\mathbf{W}} \,\|\, p_{\widehat{\mathbf{W}}}\big),
\end{aligned}$$

with

$$p_{\mathbf{W}}(\cdot) = \prod_{l=1}^{L} \prod_{g=1}^{G_l} \prod_{n=1}^{N_l} \mathcal{N}\left(\cdot \,\middle|\, \overline{I}_l(\mathbf{X}^{l-1})_{g,n} \overline{P}_l(\mathbf{W}^l)_g, \left\| \big(\overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\mathbf{W}^l)\big) \right\|_F^2 \boldsymbol{I}\right),$$

and

$$p_{\widehat{\mathbf{W}}}(\cdot) = \prod_{l=1}^{L} \prod_{g=1}^{G_l} \prod_{n=1}^{N_l} \mathcal{N}\left(\cdot \,\middle|\, \overline{I}_l(\mathbf{X}^{l-1})_{g,n} \overline{P}_l(\widehat{\mathbf{W}}^l)_g, \left\| \left(\overline{I}_l(\mathbf{X}^{l-1}) \overline{P}_l(\mathbf{W}^l) \right) \right\|_F^2 \boldsymbol{I} \right).$$

There are two main differences. First, BALF also works under the Gaussian assumption, but with the additional nuance that the variances depend on the per-layer output magnitude (in terms of the squared Frobenius norm). Second, the optimization problem is not only concerned with the low-rank projection scheme given a fixed rank per layer; the selection of ranks under some complexity budget is, itself, baked into the optimization problem. (Note that, strictly speaking, BALF solves the problem approximately in practice.)

We believe this point of view might be an interesting avenue for future research.

I Deferred Proofs

In this section, we provide full proofs for the results in the main text that were not covered in other sections of the appendix. Results are restated for the convenience of the reader.

I.1 Proof of Proposition 4.7

Proposition 4.7. *Assume that $f(\mathbf{X};\mathbf{W})$ is $(\overline{O}, \overline{I}, \overline{P})$-expressible. Then, the activation distortion (as defined in Equation (3)) incurred when low-rank projecting the parameters with $\mathcal{T}_P^{\text{AA}}(\cdot)$ is $\frac{N}{B}\sum_{g=1}^{G}\sum_{i=P+1}^{U} \boldsymbol{\sigma}_{g,i}^2$, where $\boldsymbol{\sigma}$ denotes the batched singular values of $\mathbf{M}^+\overline{P}(\mathbf{W})$ and U is the total number of singular values per group.*

Proof. First, observe that

$$\|\mathsf{cat}(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_K)\|_F^2 = \sum_{i=1}^{K} \|\boldsymbol{A}_i\|_F^2, \tag{4}$$

where cat denotes the operation that yields a third-order tensor from a batch of matrices (of the same shape). Therefore, for a batch of distortion matrices, we may analyze each term independently and sum their contributions.

Moreover,

$$\begin{aligned} \left\| f(\mathbf{X};\mathbf{W}) - f\left(\mathbf{X}; \mathcal{T}_P^{\text{AA}}(\mathbf{W})\right) \right\|_F &= \left\| \overline{O}\big(\overline{I}(\mathbf{X})\, \overline{P}(\mathbf{W})\big) - \overline{O}\big(\overline{I}(\mathbf{X})\, \mathcal{T}_P^{\text{AA}}\big(\overline{P}(\mathbf{W})\big)\big) \right\|_F \\ &= \left\| \overline{I}(\mathbf{X})\, \overline{P}(\mathbf{W}) - \overline{I}(\mathbf{X})\, \mathcal{T}_P^{\text{AA}}\big(\overline{P}(\mathbf{W})\big) \right\|_F, \end{aligned}$$

since $\overline{O}$ is a composition of reshapes and permutes, which preserve the Frobenius norm.

Fix a group index g (omitted from the notation below for simplicity). For reference, the dimensions of the different elements are: $\overline{I}(\mathbf{X}) \in \mathbb{R}^{N \times I}$, $\boldsymbol{M} \in \mathbb{R}^{I \times I}$, $\boldsymbol{M}^+ \in \mathbb{R}^{I \times I}$, and $\overline{P}(\mathbf{W}) \in \mathbb{R}^{I \times O}$.

When using activation-aware factorization and truncating to rank P, it follows that

$$\begin{aligned}
\ell_{(g)}^{\text{activ}}(P) &= \frac{1}{B}\left\|\overline{I}(\mathbf{X})\overline{P}(\mathbf{W}) - \overline{I}(\mathbf{X})\boldsymbol{M}\mathcal{T}_P^{\text{SVD}}\left(\boldsymbol{M}^+\overline{P}(\mathbf{W})\right)\right\|_F^2 \\
&\underset{\text{(a)}}{=} \frac{1}{B}\left\|\overline{I}(\mathbf{X})\boldsymbol{M}\left(\boldsymbol{M}^+\overline{P}(\mathbf{W}) - \mathcal{T}_P^{\text{SVD}}\left(\boldsymbol{M}^+\overline{P}(\mathbf{W})\right)\right)\right\|_F^2 \\
&= \frac{1}{B}\left\|\overline{I}(\mathbf{X})\boldsymbol{M}\boldsymbol{U}\widehat{\boldsymbol{\Sigma}}\boldsymbol{V}^\top\right\|_F^2 \\
&= \frac{1}{B}\operatorname{tr}\left(\boldsymbol{V}\widehat{\boldsymbol{\Sigma}}^\top\boldsymbol{U}^\top\boldsymbol{M}^\top\overline{I}(\mathbf{X})^\top\overline{I}(\mathbf{X})\boldsymbol{M}\boldsymbol{U}\widehat{\boldsymbol{\Sigma}}\boldsymbol{V}^\top\right) \\
&\underset{\text{(b)}}{=} \frac{N}{B}\operatorname{tr}\left(\boldsymbol{V}\widehat{\boldsymbol{\Sigma}}^\top\widehat{\boldsymbol{\Sigma}}\boldsymbol{V}^\top\right) \\
&= \frac{N}{B}\sum_{i=P+1}^{U}\boldsymbol{\sigma}_i^2,
\end{aligned}$$

where the SVD is taken of $\boldsymbol{M}^+\overline{P}(\mathbf{W})$, and $\widehat{\boldsymbol{\Sigma}}$ corresponds to the tail of $\boldsymbol{\Sigma}$ after truncating the singular values (i.e., it is equal to $\boldsymbol{\Sigma}$ but with the first P diagonal elements set to 0).

For step (a), note that $\overline{I}(\mathbf{X})\boldsymbol{M}\boldsymbol{M}^+ = \overline{I}(\mathbf{X})$ (see Appendix B.1).

For step (b), note the following. First, recall from the definition of $\boldsymbol{M}$ (see Appendix B.1) that $\boldsymbol{M}^\top\overline{I}(\mathbf{X})^\top\overline{I}(\mathbf{X})\boldsymbol{M} = N\overline{\boldsymbol{I}}_R$ (where R is the rank of $\overline{I}(\mathbf{X})$); and that $\boldsymbol{M}$ has its last $I-R$ columns set to 0, meaning $\boldsymbol{M}^+$ has its last $I-R$ rows set to 0.

As $\boldsymbol{U}\widehat{\boldsymbol{\Sigma}}\boldsymbol{V}^\top$ corresponds to the SVD tail of $\boldsymbol{M}^+\overline{P}(\mathbf{W}) \in \mathbb{R}^{I\times O}$, and $\boldsymbol{M}^+\overline{P}(\mathbf{W})$ has its last $I-R$ rows set to 0, all left singular vectors associated with nonzero singular values lie in the range of $\overline{\boldsymbol{I}}_R$. Hence, $\overline{\boldsymbol{I}}_R\boldsymbol{U}\widehat{\boldsymbol{\Sigma}} = \boldsymbol{U}\widehat{\boldsymbol{\Sigma}}$, or equivalently $\widehat{\boldsymbol{\Sigma}}^\top\boldsymbol{U}^\top\overline{\boldsymbol{I}}_R\boldsymbol{U}\widehat{\boldsymbol{\Sigma}} = \widehat{\boldsymbol{\Sigma}}^\top\widehat{\boldsymbol{\Sigma}}$. This results in $\widehat{\boldsymbol{\Sigma}}^\top\boldsymbol{U}^\top\boldsymbol{M}^\top\overline{I}(\mathbf{X})^\top\overline{I}(\mathbf{X})\boldsymbol{M}\boldsymbol{U}\widehat{\boldsymbol{\Sigma}} = N\widehat{\boldsymbol{\Sigma}}^\top\boldsymbol{U}^\top\overline{\boldsymbol{I}}_R\boldsymbol{U}\widehat{\boldsymbol{\Sigma}} = N\widehat{\boldsymbol{\Sigma}}^\top\widehat{\boldsymbol{\Sigma}}$.

From the observation in Equation (4), it follows that the total distortion is the sum across each group's distortion

$$\ell^{\text{activ}}(P) = \sum_{g=1}^{G}\ell_{(g)}^{\text{activ}}(P) = \frac{N}{B}\sum_{g=1}^{G}\sum_{i=P+1}^{U}\boldsymbol{\sigma}_{g,i}^2,$$

which completes the proof.

□

I.2 Proof of Proposition 4.6

Lemma I.1. *Let $\boldsymbol{Y} \in \mathbb{R}^{M\times N}$ satisfy $\boldsymbol{Y}^\top\boldsymbol{Y} = \alpha\boldsymbol{I}_N$ with $\alpha > 0$. Then, for any $\boldsymbol{X} \in \mathbb{R}^{N\times K}$,*

$$\boldsymbol{\sigma}_i(\boldsymbol{Y}\boldsymbol{X}) = \sqrt{\alpha}\boldsymbol{\sigma}_i(\boldsymbol{X})$$

for all i, with singular values padded by zeros if necessary. Consequently, for any $P \geq 1$,

$$\sum_{i=1}^{P}\boldsymbol{\sigma}_i^2(\boldsymbol{Y}\boldsymbol{X}) = \alpha\sum_{i=1}^{P}\boldsymbol{\sigma}_i^2(\boldsymbol{X}).$$

Proof. We have

$$(\boldsymbol{Y}\boldsymbol{X})^\top(\boldsymbol{Y}\boldsymbol{X}) = \boldsymbol{X}^\top(\boldsymbol{Y}^\top\boldsymbol{Y})\boldsymbol{X} = \alpha\boldsymbol{X}^\top\boldsymbol{X}.$$

Hence, the eigenvalues of $(\boldsymbol{Y}\boldsymbol{X})^\top(\boldsymbol{Y}\boldsymbol{X})$ are exactly α times those of $\boldsymbol{X}^\top\boldsymbol{X}$, including zeros. Therefore, $\boldsymbol{\sigma}_i(\boldsymbol{Y}\boldsymbol{X}) = \sqrt{\alpha}\boldsymbol{\sigma}_i(\boldsymbol{X})$ for all i. Squaring and summing the first P singular values gives the claim. □

Proposition 4.6. *$\mathcal{T}_P^{\text{AA}}(\cdot)$ is optimal in the sense of Definition 4.5.*

Proof. It suffices to prove the result for a single group. Fix a group (its index is omitted from the notation for brevity), and let $\boldsymbol{W} = \overline{P}(\mathbf{W}) \in \mathbb{R}^{I \times O}$, $\boldsymbol{X} = \overline{I}(\mathbf{X}) \in \mathbb{R}^{N \times I}$, $\boldsymbol{M} \in \mathbb{R}^{I \times I}$, and $\boldsymbol{M}^+ \in \mathbb{R}^{I \times I}$ denote the items corresponding to said group, and denote the rank of $\boldsymbol{X}$ by R.

We recall the following facts; see Appendix B.1. First, $\boldsymbol{M}$ has its last $I - R$ columns set to 0, which forces $\boldsymbol{M}^+$ to have its last $I - R$ rows set to 0. Hence, $\boldsymbol{M}\boldsymbol{M}^+ = \boldsymbol{M}\boldsymbol{I}_{:,:R}\boldsymbol{I}_{:R,:}\boldsymbol{M}^+$. Moreover, from the definition of $\boldsymbol{M}$, we have $\boldsymbol{M}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{M} = N\overline{\boldsymbol{I}}_R$, and therefore, it holds that $\boldsymbol{I}_{:R,:}\boldsymbol{M}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{M}\boldsymbol{I}_{:,:R} = N\boldsymbol{I}_R$. It is also true that $\boldsymbol{X}\boldsymbol{M}\boldsymbol{M}^+ = \boldsymbol{X}$.

It follows that

$$\sum_{i=1}^{P} \boldsymbol{\sigma}_i^2(\boldsymbol{X}\boldsymbol{W}) = \sum_{i=1}^{P} \boldsymbol{\sigma}_i^2(\boldsymbol{X}\boldsymbol{M}\boldsymbol{M}^+\boldsymbol{W}) = \sum_{i=1}^{P} \boldsymbol{\sigma}_i^2(\boldsymbol{X}\boldsymbol{M}\boldsymbol{I}_{:,:R}\boldsymbol{I}_{:R,:}\boldsymbol{M}^+\boldsymbol{W}) = N\sum_{i=1}^{P} \boldsymbol{\sigma}_i^2(\boldsymbol{M}^+\boldsymbol{W}),$$

where the last equality follows from Lemma I.1.

Hence,

$$\begin{aligned}
\|\overline{I}(\mathbf{X})\overline{P}(\mathbf{W}) - \mathcal{T}_P^{\mathrm{SVD}}\big(\overline{I}(\mathbf{X})\overline{P}(\mathbf{W})\big)\|_F^2 &= \sum_{i=P+1}^{U} \boldsymbol{\sigma}_i^2(\boldsymbol{X}\boldsymbol{W}) \\
&= N\sum_{i=P+1}^{U} \boldsymbol{\sigma}_i^2(\boldsymbol{M}^+\boldsymbol{W}) \\
&= \|\overline{I}(\mathbf{X})\overline{P}(\mathbf{W}) - \overline{I}(\mathbf{X})\mathcal{T}_P^{\mathrm{AA}}\big(\overline{P}(\mathbf{W})\big)\|_F^2,
\end{aligned}$$

where the last equality follows from Proposition 4.7.

Taking square roots completes the proof.

□

J LLM Usage Disclosure

In this section, we disclose the role of LLMs in this work. We used LLMs to help identify related work, including potential comparison baselines, in addition to manual search. All cited works and reported comparisons were read by the authors; LLMs primarily assisted with search. We also used LLMs to discuss research ideas, assist with technical claims, support code development, and proofread and polish the writing.