Fisher-Orthogonal Projection Methods for Natural Gradient Descent with Large Batches

Yishun Lu [1]**, Wesley Armour** [1]

[1]Department of Engineering Science University of Oxford Oxford, UK

Abstract

Modern GPUs are equipped with large amounts of high-bandwidth memory, enabling them to support mini-batch sizes of up to tens of thousands of training samples. However, most existing optimizers struggle to perform effectively at such a large batch size. As batch size increases, gradient noise decreases due to averaging over many samples, limiting the ability of first-order methods to escape sharp or suboptimal minima and reach the global minimum. Meanwhile, second-order methods like the natural gradient with Kronecker-Factored Approximate Curvature (KFAC) often require excessively high damping to remain stable at large batch sizes. This high damping effectively "washes out" the curvature information that gives these methods their advantage, reducing their performance to that of simple gradient descent. In this paper, we introduce Fisher-Orthogonal Projection (FOP), a novel technique that restores the effectiveness of the second-order method at very large batch sizes, enabling scalable training with improved generalization and faster convergence. FOP constructs a variance-aware update direction by leveraging gradients from two sub-batches, enhancing the average gradient with a component of the gradient difference that is orthogonal to the average under the Fisher-metric. Through extensive benchmarks, we show that FOP accelerates convergence by $\times 1.2$–1.3 over KFAC and $\times 1.5$–1.7 over SGD/AdamW at the same moderate batch sizes, while at extreme scales it achieves up to a $\times 7.5$ speedup. Unlike other methods, FOP maintains small-batch accuracy when scaling to extremely large batch sizes. Moreover, it reduces Top-1 error by 2.3–3.3% on long-tailed CIFAR benchmarks, demonstrating robust generalization under severe class imbalance. Our lightweight, geometry-aware use of intra-batch variance makes natural-gradient optimization practical on modern data-centre GPUs. FOP is open-source and pip-installable, which can be integrated into existing training code with a single line and no extra configuration.

Introduction

The increasing scale of modern language models and vision transformers has made large mini-batch training a necessity. Modern GPUs offer extensive high-bandwidth memory (such as 192GB in AMD MI300X), and data centers often combine hundreds of these devices, enabling the efficient processing of tens of thousands of training examples per batch. This improves hardware utilization, reduces communication overhead, and accelerates training significantly. As batch sizes increase, the gradients become more deterministic. This reduces the stochastic noise that previously helped first-order optimizers such as SGD (Robbins and Monro 1951), Adam (Kingma and Ba 2014), and AdamW (Loshchilov and Hutter 2017) to explore flatter minima of the loss landscape. The loss of this noise (Keskar et al. 2016; McCandlish et al. 2018) requires first-order methods to rely on smaller learning rates and stronger explicit regularization to maintain stability and generalization performance.

Natural-Gradient Descent (NGD) addresses scenarios where stochastic gradient noise is minimal, such as very large mini-batches (Pascanu and Bengio 2013; Shazeer and Stern 2018; Ishikawa and Yokota 2024). Unlike first-order methods, NGD incorporates second-order curvature information via the Fisher information matrix (Amari 1998), enabling geometry-aware parameter updates invariant to model parameterization. However, exact computation of the Fisher matrix is infeasible for modern neural networks. Practical implementations rely on approximations, with Kronecker-Factored Approximate Curvature (KFAC) being the most widely adopted (Martens and Grosse 2015). KFAC simplifies the Fisher matrix by exploiting the layer-wise structure of neural networks, making the approximation computationally efficient. Subsequent methods further approximate KFAC to improve tractability (Lin et al. 2024; Yang et al. 2020; Liu et al. 2024; Eschenhagen et al. 2023). Despite its promise, KFAC struggles in the very-large-batch regime required for modern hardware. As the batch size grows, the Fisher matrix becomes increasingly ill-conditioned (Sagun, Bottou, and LeCun 2016; Ghorbani, Krishnan, and Xiao 2019), leading to numerical instability. This forces the use of strong damping, which unfortunately suppresses the very curvature information that gives KFAC its advantage.

Prior attempts to scale natural-gradient methods to large-batch training include SENG (Yang et al. 2020), which uses low-rank sketches but introducing new hyperparameters, and SP-NGD (Osawa et al. 2020), which relies on empirical-Fisher approximations and stale-statistic heuristics that require task-specific hyperparameter retuning. Adaptive batch-size schedules (Yao et al. 2018) improve throughput but still rely on heavily damped mean gradients. Although these methods reduce hardware require-

ments, their update rules remain fundamentally unchanged, still dominated by mean gradients and extensive hyperparameter tuning.

In this paper, we propose Fisher-Orthogonal Projection (FOP), which augments natural gradient descent with a Fisher-orthogonal variance correction. This novel geometry-aware update captures intra-batch gradient variation without relying on sketching ranks, stale-statistics heuristics, or customized communication strategies. Specifically, our contributions include:

- **Fisher–Orthogonal Projection optimizer.** We propose a novel second-order update that augments the natural gradient with a geometry-aware, variance-controlled component, capturing intra-batch information by standard KFAC.

- **Extreme large-batch scalability.** We demonstrate that FOP seamlessly scales to cases where SGD, AdamW, and K-FAC break down, and it can achieve speedups of up to ×7.5 in wall-clock time while maintaining convergence at extremely large batch sizes in ImageNet and CIFAR datasets.

- **Robust generalization under imbalance.** Reducing Top-1 error rate by 2.3–3.3% on long-tailed CIFAR-LT benchmarks without additional tricks

- **Distributed FOP.** Efficiently sharding Fisher computation across GPUs with dual-gradient AllReduce and broadcasted updates, enabling scalable, low-overhead training.

Natural Gradient Descent

Natural Gradient Descent is a second-order optimization method derived from Newton's method, where the Hessian is replaced by the Fisher Information Matrix to reflect the geometry of the parameter space. In standard gradient descent, the update rule is:

$$\theta_i = \theta_{i-1} - \eta \nabla_\theta \mathcal{L}(\theta_{i-1})$$

where θ_i is the parameter vector at iteration i, η is the learning rate, and $\nabla_\theta \mathcal{L}(\theta_{i-1})$ is the gradient of the loss function $\mathcal{L}$ evaluated at θ_{i-1}. This treats all directions in parameter space uniformly, which can lead to slow convergence in ill-conditioned landscapes. Newton's method improves this by preconditioning the gradient with the inverse Hessian H^{-1}, but computing the full Hessian is often infeasible for large models.

Natural Gradient Descent modifies the update by replacing the Hessian with the Fisher Information Matrix F, which defines a Riemannian metric on the statistical manifold (Amari 1998):

$$\theta_i = \theta_{i-1} - \eta F^{-1} \nabla_\theta \mathcal{L}(\theta_{i-1})$$

This yields the steepest descent direction under the Kullback–Leibler (KL) divergence, making the update invariant to parameter reparameterizations (Martens and Grosse 2015).
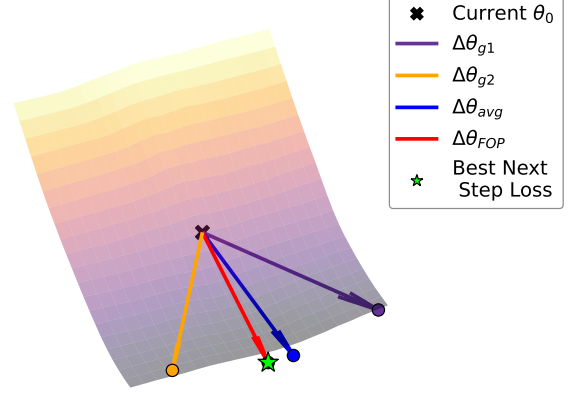

Figure 1: 3D loss landscape of ResNet-18 with CIFAR-10 for batch size of 1024. Arrows represent the direction of the steps of different gradients. The green star is the smallest loss after updating the model based on different update directions.

Fisher-Orthogonal Projection

A loss landscape is plotted based on the method suggested in (Li et al. 2018) in Figure 1, by projecting the high-dimensional parameter space onto a 2D plane using two random but normalized directions and evaluating the loss across a grid in this plane. Naively averaging gradients across mini-batches can obscure useful optimization directions due to averaging over many samples. In particular, such averaging may suppress informative signals when gradients from different batches point in significantly different directions. Especially at large batch sizes, where gradient variability is low but intra-batch differences can still carry important optimization signals. We propose the *Fisher-Orthogonal Projection (FOP)* method, which preserves the informative structure of each mini-batch gradient while ensuring stable and curvature-aware descent. The key idea is to use the average gradient as the primary descent direction, capturing the common signal shared across mini-batches. In addition, FOP introduces a Fisher-orthogonal component, derived from the difference between two mini-batch gradients. This orthogonal component contains complementary curvature-sensitive information that would otherwise be lost through simple averaging.

Suppose that at a parameter point θ, we compute two gradients g_1 and g_2 from two independent mini-batches. Then the FOP update is defined as:

$$L_1(\theta), \quad L_2(\theta) \tag{1}$$

each associated with a gradient:

$$g_1 = \nabla_\theta L_1(\theta), \quad g_2 = \nabla_\theta L_2(\theta) \tag{2}$$

We compute the average and difference of the two gradients:

$$g_{\text{avg}} = \frac{1}{2}(g_1 + g_2), \quad g_{\text{diff}} = g_1 - g_2. \tag{3}$$

To eliminate redundancy and extract only novel information, we orthogonalize g_{diff} with respect to g_{avg} under the inner product induced by the Fisher matrix F. The projection scalar is computed as:

$$s_{\text{proj}} = \frac{g_{\text{diff}}^{\top} F g_{\text{avg}}}{g_{\text{avg}}^{\top} F g_{\text{avg}} + \epsilon} \tag{4}$$

and the orthogonal component is then:

$$g_{\text{diff}}^{\perp} = g_{\text{diff}} - s_{\text{proj}} \cdot g_{\text{avg}}. \tag{5}$$

By construction, this ensures that $\langle g_{\text{avg}}, g_{\text{diff}}^{\perp} \rangle_F = 0$ (as described in Lemma 1 in the Appendix), meaning the new component contains only the information that is orthogonal in the Fisher geometry, which is already captured in the average gradient.

The final combined update direction is given by:

$$g_{\text{combined}} = g_{\text{avg}} + \beta g_{\text{diff}}^{\perp}, \tag{6}$$

where β is a scalar weight, adaptively determined to locally minimize the primary or total loss. The overall parameter update using Natural Gradient Descent then becomes:

$$\theta_{t+1} = \theta_t - \eta F^{-1} g_{\text{combined}}. \tag{7}$$

Layer-wise Adaptive Coefficient β

When the true optimization objective is the sum of two per-batch losses, we can write the total loss as:

$$L_{\text{tot}}(\theta) = L_1(\theta) + L_2(\theta). \tag{8}$$

To minimize this objective locally, we seek an optimal mixing coefficient β that balances the direction $g_{\text{diff}}^{\perp}$ relative to the average gradient. We derive this optimal β (denoted β^*) using a second-order Taylor approximation.

We begin with the natural-gradient update step of the form as in Eq. 7:

$$\theta_{\text{new}} = \theta - \eta F^{-1}(g_{\text{avg}} + \beta g_{\text{diff}}^{\perp}), \tag{9}$$

Using a second-order Taylor expansion of L_{tot} around θ, and assuming the Hessian matrix approximates to the Fisher matrix ($\nabla^2 L_{\text{tot}} \approx F$), the approximate loss after update is:

$$L_{\text{tot}}(\theta - \eta d) \approx L_{\text{tot}}(\theta) - \eta(g_1 + g_2)^{\top} d + \frac{\eta^2}{2} d^{\top} F d, \tag{10}$$

where

$$d = F^{-1}(g_{\text{avg}} + \beta g_{\text{diff}}^{\perp}). \tag{11}$$

To isolate the effect of β, we define a surrogate objective $J_{\text{tot}}(\beta)$ by dropping constants and factors of η:

$$J_{\text{tot}}(\beta) = -(g_1 + g_2)^{\top} F^{-1}(g_{\text{avg}} + \beta g_{\text{diff}}^{\perp})$$
$$+ \frac{1}{2}(g_{\text{avg}} + \beta g_{\text{diff}}^{\perp})^{\top} F^{-1}(g_{\text{avg}} + \beta g_{\text{diff}}^{\perp}). \tag{12}$$

To simplify, we define the following inner products:

$$D = g_{\text{avg}}^{\top} F^{-1} g_{\text{diff}}^{\perp}, \quad E = (g_{\text{diff}}^{\perp})^{\top} F^{-1} g_{\text{diff}}^{\perp}. \tag{13}$$

Noting that $g_1 + g_2 = 2g_{\text{avg}}$, we substitute into (12) and expand:

$$J_{\text{tot}}(\beta) = -2D - 2\beta D + \frac{1}{2}\left(\|g_{\text{avg}}\|_{F^{-1}}^2 + 2\beta D + \beta^2 E\right)$$
$$= -2D - \beta D + \frac{1}{2}\beta^2 E + \text{const}, \tag{14}$$

where $\|g_{\text{avg}}\|_{F^{-1}}^2 = g_{\text{avg}}^{\top} F^{-1} g_{\text{avg}}$ is absorbed into the constant term.

To find the optimal β^*, we differentiate (14) with respect to β and set the derivative to zero:

$$\frac{dJ_{\text{tot}}}{d\beta} = -D + \beta^* E = 0 \quad \Rightarrow \quad \beta^* = \frac{D}{E}. \tag{15}$$

Substituting back the definitions of D and E, we obtain the closed-form expression:

$$\beta^* = \frac{g_{\text{avg}}^{\top} F^{-1} g_{\text{diff}}^{\perp}}{(g_{\text{diff}}^{\perp})^{\top} F^{-1} g_{\text{diff}}^{\perp}}. \tag{16}$$

This yields a layer-wise coefficient β^* that minimizes our second-order surrogate, injecting orthogonal corrections only when beneficial. While it relies on the Hessian–Fisher approximation, which can misestimate curvature early in training or in highly nonlinear models, damped Fisher matrices and large-batch averaging curb these errors. In practice, any misleading orthogonal signal drives $\beta^* \to 0$, safely reducing FOP to a standard KFAC update.

Layer-wise Adaptive Scaling Step Size η_{ℓ}^*

To ensure that each layer's update magnitude is automatically adjusted to account for its local curvature and gradient alignment, we introduce a layer-wise adaptive coefficient η_{ℓ}^*. Rather than using a single global learning rate for all parameters, η_{ℓ}^* is chosen to (locally) minimize a quadratic approximation of the loss function along the natural-gradient direction for each layer ℓ. Instead of differentiating about β in Eq. 10, we minimize this one-dimensional quadratic model in η. After we set the derivative with respect to η to zero, the optimal step size becomes:

$$\eta_{\ell}^* = \frac{g_{\ell,\text{tot}}^{\top} F_{\ell}^{-1} g_{\ell,\text{comb}}}{g_{\ell,\text{comb}}^{\top} F_{\ell}^{-1} g_{\ell,\text{comb}}}. \tag{17}$$

This expression automatically adjusts the step size based on both the alignment between the total gradient and the proposed update direction, and the curvature in that direction. When the curvature along $g_{\ell,\text{comb}}$ is low and the update is well-aligned with the descent direction, η_{ℓ}^* will approach 1, allowing a full natural-gradient step. Conversely, when the curvature is high or the combined gradient is poorly aligned with the total gradient, η_{ℓ}^* decreases below 1, effectively damping the update to avoid overshooting. The final updates are:

$$d_{\ell} = \eta_0 \eta_{\ell}^* F_{\ell}^{-1} g_{\ell,\text{comb}}, \tag{18}$$

where η_0 is the base learning rate shared across the whole model.

Kullback–Leibler (KL) norm analysis

Natural-gradient methods, such as KFAC (Martens and Grosse 2015), select update directions that maximize progress in parameter space relative to the KL-divergence between the model before and after the update. A standard second-order approximation of the KL-divergence gives rise to the KL-norm:

$$\text{KL}(p_{\theta+\Delta}\|p_\theta) \approx \frac{1}{2}\|F^{1/2}\Delta\|^2 \qquad (19)$$

where F is the Fisher information matrix and Δ is the parameter update step. The KL-norm quantifies how much the model distribution changes due to an update.

In our case, the FOP update step is defined as:

$$\Delta_{\text{FOP}} = -\eta M^{-1}(g + \beta g^\perp) \qquad (20)$$

where g is the average micro-batch gradient g_{avg}, $g^\perp$ is an orthogonal correction term in Eq. 6, satisfying $(g^\perp)^\top Fg = 0$, $M = F + \lambda I$ is the damped Fisher matrix, and $\beta \in \mathbb{R}$ controls the contribution of the orthogonal component.

Substituting this into the KL-norm expression we obtain a decomposition into three terms:

$$\left\|F^{1/2}\Delta_{\text{FOP}}\right\|^2 = \eta^2\left[g^\top Qg + 2\beta g^\top Qg^\perp + \beta^2(g^\perp)^\top Qg^\perp\right] \qquad (21)$$

- A base-term $g^\top Qg$ corresponding to standard KFAC,
- A cross-term $2\beta\, g^\top Qg^\perp$,
- And an orthogonal-term $\beta^2(g^\perp)^\top Qg^\perp$,

where $Q = (F + \lambda I)^{-1}F(F + \lambda I)^{-1}$ acts as a curvature–weighted metric. In the large-damping case, where $\lambda \gg \Lambda_i$ for every Fisher eigenvalue Λ_i that carries weight in g, the spectral factor $\frac{\Lambda_i}{(\Lambda_i+\lambda)^2}$ is proportional to Λ_i/λ^2. Therefore, the base term scales as $g^\top Qg \propto \lambda^{-2}$. As shown in Appendix B, the two additional terms from FOP decay even more gently. Specifically:

$$\|F^{1/2}\Delta_{\text{FOP}}\|^2 \leq \eta^2 \Bigg[\underbrace{g^\top Qg}_{\|F^{1/2}\Delta_{\text{KFAC}}\|^2}$$
$$+ 2\beta\frac{\mu_{\max}}{\lambda}\|F^{-1/2}g\| \cdot \|F^{-1/2}g_\perp\|$$
$$+ \beta^2\frac{\mu_{\max}}{4\lambda}\|F^{-1/2}g_\perp\|^2 \Bigg] \qquad (22)$$

Because the KL-norm of the FOP update splits into a base-term that decays as $\mathcal{O}(1/\lambda^2)$ and two correction terms that decay only as $\mathcal{O}(1/\lambda)$, there is an inherent separation in how quickly these components diminish as the damping parameter λ increases.

During the early phase of training, it is common for the $\|g_{\text{avg}}\|$ to be large, while the $\|g_{\text{diff}\perp}\|$ is also notable and dominated by high-frequency, low-curvature noise that is exaggerated by the application of F^{-1}. In such scenarios, the orthogonal correction $g_{\text{diff}}^\perp$ often points in the opposite direction to the main descent path. As a result, the optimal mixing

coefficient becomes negative ($\beta < 0$), leading to a negative cross-term in the KL-norm. This partial cancellation of the core component creates margins that safely reduce the damping factor λ.

Distributed FOP

Algorithm 1: Distributed FOP with Dual Gradients

Require:
    M : neural network model
    $P = \{p_0, \ldots, p_{N-1}\}$: set of N GPU processes
    S_j : subset of layers for which p_j is the curvature *specialist*
    $\mathcal{D}_j$: local mini-batch on p_j
    F_i : running estimate of the Fisher matrix block for layer ℓ_i, stored and updated by its specialist GPU p_k

    $G_{\text{pri}} = \{p_j \in P \mid j \bmod 2 = 0\}$ ▷ primary group
    $G_{\text{sec}} = \{p_j \in P \mid j \bmod 2 = 1\}$ ▷ secondary group

Ensure: preconditioned global gradient $\tilde{g}$
1: **for all** $p_j \in P$ **in parallel do**
2:     $g_j \leftarrow \nabla_\theta \mathcal{L}(M; \mathcal{D}_j)$ ▷ local back-prop
3:     **if** curvature update step **then**
4:         **for all** layers ℓ_i in M **do**
5:             $p_k \leftarrow$ specialist s.t. $\ell_i \in S_k$
6:             send local curvature factors of ℓ_i to p_k
                        /* async, non-blocking */
7:             **if** $j = k$ **then**
8:                 update and invert $F_i \rightarrow F_i^{-1}$
9:             **end if**
10:         **end for**
11:     **end if**
12:     **if** $p_j \in G_{\text{pri}}$ **then**
13:         global ALLREDUCE to compute g_1
14:     **else**
15:         global ALLREDUCE to compute g_2
16:     **end if**
17:     **for all** layers ℓ_i in M **do**
18:         $p_k \leftarrow$ specialist for ℓ_i
19:         **if** $j = k$ **then**
20:             $\tilde{g}_i \leftarrow \text{FOP}(g_{1,i}, g_{2,i}, F_i^{-1})$
21:             broadcast $\tilde{g}_i$ to all processes
22:         **end if**
23:     **end for**
24: **end for**
25: assemble $\tilde{g} \leftarrow [\tilde{g}_1, \ldots, \tilde{g}_L]$ ▷ L = #layers

---

Traditional second-order optimization methods like KFAC are notoriously difficult to scale due to the high memory and computation costs of storing and inverting large curvature matrices. Moreover, synchronizing these matrices across multiple GPUs introduces significant communication overhead, making them impractical for large-scale training without careful system design. We design a scalable FOP implementation that combines data parallelism with lightweight model parallelism to minimize the overhead of splitting large batches. First, we assign each GPU as a specialist, responsible for updating the curvature (Fisher infor-
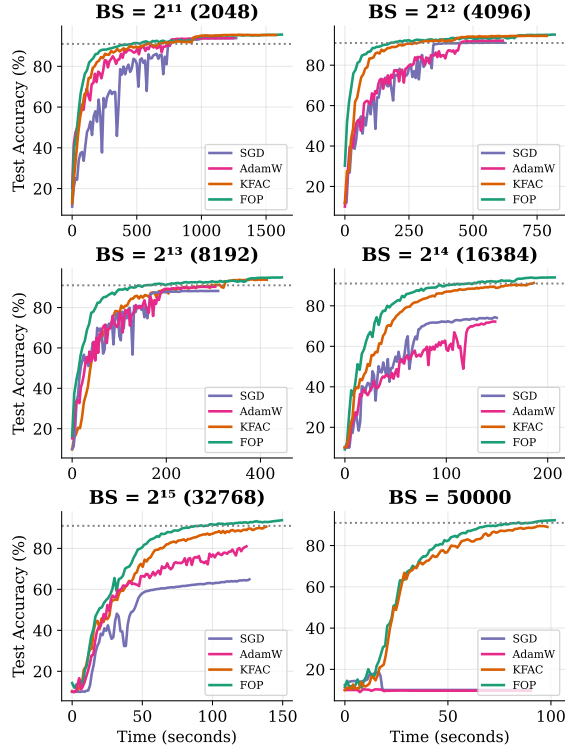
Figure 2: Test accuracy vs. wall-clock time (in seconds) for ResNet-18 on CIFAR-10, grouped by batch size. The dotted line represents the threshold of 91%.

mation) of a subset of layers via a sharded preconditioner similar to the previous works (Osawa et al. 2023; Pauloski et al. 2022). Second, we introduce a dual-gradient reduction strategy, where two global gradients $\mathbf{g}_1$ and $\mathbf{g}_2$ are computed in parallel over disjoint GPU groups. Finally, each specialist GPU applies the FOP using its local Fisher inverse and both gradients to compute its layer's update, which is then broadcast across the GPUs. This distributed design enables efficient second-order updates across large-scale multi-GPU systems. The full algorithm is shown in Algorithm 1.

Experiments

In this section, we rigorously evaluate FOP against both first-order (SGD, AdamW) and second-order (KFAC) baselines across four vision benchmarks: CIFAR-10 with ResNet-18, ImageNet-100 with T2T-ViT, ImageNet-1K with ResNet-50, and long-tailed CIFAR10-LT/100-LT with ResNet-32, demonstrating its fast convergence, large-batch scalability, and robustness under class imbalance. To ensure fair and rigorous comparisons, we evaluate all methods on several standard benchmarks: ResNet-18 on CIFAR-10, ResNet-50 on ImageNet-1k, and a Vision Transformer (ViT) on ImageNet-100. For each setting, we perform an extensive hyperparameter search for all optimizers. This includes tuning the learning rate, and for second-order methods like KFAC and FOP, also tuning the damping ratio λ. Following the linear scaling rule from (Goyal et al. 2017), the learning

rate is scaled proportionally with the batch size. Additionally, the curvature update frequency for the second-order optimizer is reduced as the batch size increases, until it reaches a minimum threshold of 5 steps. To isolate the effect of our Fisher-orthogonal projection, FOP and KFAC share identical learning-rate schedules in every experiment. While our results highlight FOP's advantages, we do not claim that FOP is a universally superior optimizer for all tasks and architectures. Rather, the evidence shows that augmenting natural gradient methods with a principled, geometry-aware variance component offers via FOP a robust and scalable path for second-order optimization in modern large-batch training scenarios. All experiments were performed on a single node with two AMD EPYC 9534 64-core CPUs and eight AMD MI300X GPUs. The implementations of KFAC and FOP rely on the ASDL package (Osawa et al. 2023). Full details of the hyperparameter search space, such as learning rate, damping rate, and random seeding number, and a discussion about the memory overhead are provided in the Appendix.

CIFAR10 with ResNet18

We first evaluate optimization performance on the CIFAR-10 dataset (Krizhevsky, Hinton et al. 2009), a widely used benchmark for assessing second-order optimizers (Eschenhagen et al. 2023; Liu et al. 2024; Martens and Grosse 2015). Each experiment is run with 5 different random seeds to ensure robustness. We employ the `ReduceLROnPlateau` learning rate scheduler during training. Figure 2 illustrates the progression of test accuracy over wall-clock time for ResNet-18, across batch sizes ranging from 2048 to 50000 (the total number of training samples in CIFAR-10). At small to moderate batch sizes (e.g., 2048 and 4096), all optimizers, including SGD, AdamW, KFAC, and FOP, achieve 91% accuracy, though FOP consistently reaches this threshold the fastest. As batch size increases beyond 16384, first-order optimizers such as SGD and AdamW struggle to converge within the same epoch limit, failing to reach 91% accuracy altogether at larger batch sizes (e.g., 32768 and 50000).

These trends are quantitatively summarized in Table 1, which reports both the epochs and total wall-clock time to reach 91% Top-1 accuracy on CIFAR-10 using the same GPU count. At BS=2048, FOP hits the target accuracy in 29/475.2s, with ×1.56 faster than SGD (58/743.3s) and ×1.26 faster than KFAC (37/588.7s). As we scale to BS=4096 and 8192, FOP's speedup over SGD grows to ×1.69 and ×2.91, respectively, and reaches ×3.78 at BS=16384. Crucially, FOP is the only method to arrive 91% at BS=32768 and 50000, doing so in 60/90.6s (×5.05) and 82/84.3s (×5.43). These results underscore FOP's exceptional large-batch scalability and its ability to deliver substantial accelerations.

ImageNet-100 with T2T-ViT

To evaluate FOP on modern transformers, we train a Tokens-to-Token Vision Transformer (T2T-ViT) from scratch on ImageNet-100 with running 3 different random seeds(Yuan et al. 2021; Lin et al. 2024). Following Lin et al. (2024), we

Batch Size	Epochs / Time (s) and Speedup			
(GPU)	SGD	AdamW	KFAC	FOP (ours)
2048 (2)	58 / 743.3	61 / 768.4	37 / 588.7	29 / 475.2
4096 (2)	73 / 457.9 –	73 / 454.0 –	34 / 270.5 [×1.69]	22 / 181.9 [×2.52]
8192 (2)	– / – –	– / – –	71 / 296.4 [×1.54]	35 / 157.5 [×2.91]
16384 (2)	– / – –	– / – –	99 / 186.4 [×2.46]	58 / 121.2 [×3.78]
32768 (2)	– / – –	– / – –	– / – –	60 / 90.6 [×5.05]
50000 (2)	– / – –	– / – –	– / – –	82 / 84.3 [×5.43]

Table 1: Epoch and training time (in seconds) to reach 91% Top-1 accuracy on CIFAR-10. Each row shows the batch size and number of GPUs used in the format: *Batch (GPU)* and the corresponding *Epoch/Time*. "–" indicates the accuracy threshold was not reached. For KFAC and FOP, the bracketed numbers show the speedup factor relative to SGD at the batch size of 4096.

apply FOP and KFAC only to the convolutional and linear layers' gradients and activations, while all other parameters (e.g., LayerNorm) are updated with AdamW (Loshchilov and Hutter 2017). We run 100 epochs with a cosine learning-rate schedule and measure Top-1 accuracy over wall-clock time (Figure 3). We set our target at 80.6%, the best result achieved by AdamW at batch size 512, and report the epochs and training time to reach it in Table 2.

AdamW requires nearly the full 100 epochs, and the longest runtime at batch size 512, whereas both second-order methods hit 80.6% in substantially fewer epochs and less time. FOP consistently outperforms KFAC and AdamW across batch sizes larger than 512, delivering speedups of ×4.33, ×6.90, and ×10.48, where KFAC only achieves ×3.80, ×6.34, and ×6.45 compared to AdamW. KFAC scales better than AdamW but still lags behind FOP, typically needing more epochs to match the same accuracy.

Batch Size	Epochs / Time (s) and Speedup		
(GPU)	AdamW	KFAC	FOP (ours)
512 (1)	97 / 17 498.8	42 / 8315.6 [×2.10]	44 / 9535.7 [×1.84]
1024 (2)	–	49 / 4603.6 [×3.80]	41 / 4038.9 [×4.33]
2048 (4)	–	57 / 2760.5 [×6.34]	48 / 2535.4 [×6.90]
4096 (8)	–	87 / 2715.3 [×6.45]	49 / 1670.4 [×10.48]

Table 2: Epoch and training time (in seconds) to reach 80.6% Top-1 accuracy for T2T-ViT on ImageNet-100. For KFAC and FOP, the bracketed numbers show the speedup factor relative to AdamW at the batch size of 512.
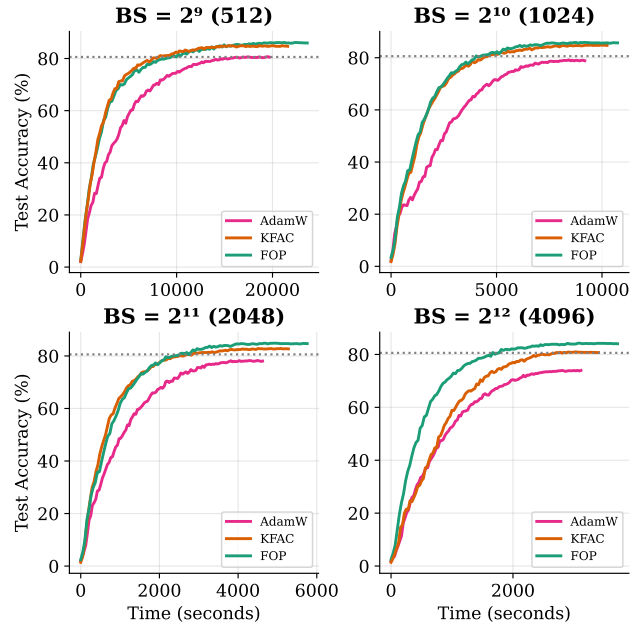

Figure 3: Test accuracy vs. wall-clock time (in seconds) for T2T-ViT on ImageNet-100, grouped by batch size. The dotted line represents the threshold of 80.6%.

ImageNet-1K with ResNet50

Batch Size	Epochs / Time (min) and Speedup		
(GPU)	SGD	KFAC	FOP (ours)
1024 (1)	71 / 2511.1	35 / 1336.5 [×1.88]	32 / 1306.1 [×1.92]
2048 (2)	– / –	– / –	33 / 999.5 [×2.51]
4096 (4)	– / –	– / –	34 / 514.9 [×4.88]
8192 (8)	– / –	– / –	40 / 335.1 [×7.50]

Table 3: Epoch and training time (in minutes) to reach 75.9% Top-1 accuracy on ImageNet-1K with ResNet-50. For KFAC and FOP, the bracketed numbers show the speedup factor relative to SGD at the batch size of 1024.

To evaluate FOP at a much larger-scale dataset, we train ResNet-50 from scratch on ImageNet-1K (Deng et al. 2009) with running 3 different random seeds, to see if it can reach Top-1 test accuracy of 75.9%, which is a standard large-scale convolutional benchmark (Mattson et al. 2019). Following Yang et al. (2020); Liu et al. (2024), we run SGD for 80 epochs with a cosine learning-rate schedule, and we train both KFAC and FOP for 50 epochs using an exponential schedule on their learning rates.

Figure 4 and Table 3 illustrate the dramatic efficiency gains of FOP in reaching 75.9% Top-1 accuracy on ImageNet-1K. At BS=1024, FOP converges in 32/1306.1 min, that is nearly ×2 faster than SGD (71/2511.1 min) and just slightly ahead

BS = 2¹⁰ (1024) | **BS = 2¹¹ (2048)**
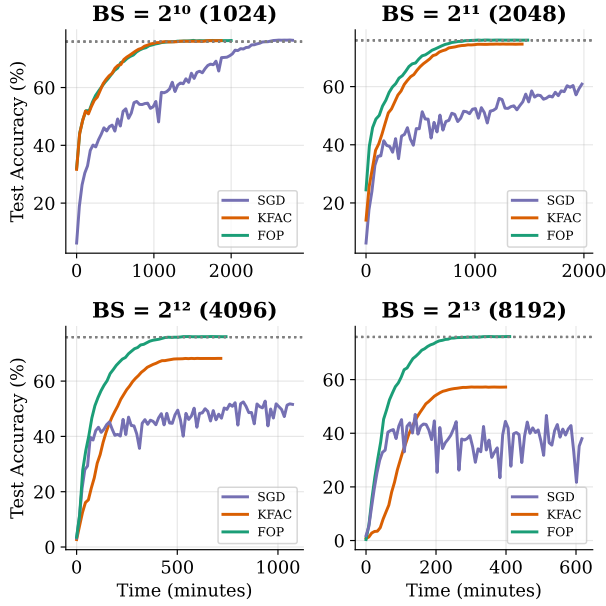**BS = 2¹² (4096)** | **BS = 2¹³ (8192)**

Figure 4: Test accuracy vs. wall-clock time (in minutes) for ResNet-50 on ImageNet-1K, grouped by batch size. The dotted line represents the threshold of 75.9%.

of K-FAC's 35/1336.5 min ($\times 1.88$). Beyond this scale, only FOP succeeds to hit the threshold in 33/999.5 min at BS=2048 ($\times 2.51$), 34/514.9 min at BS=4096 ($\times 4.88$), and 40/335.1 min at BS=8192 ($\times 7.50$), while both SGD and K-FAC stall below 75.9%. FOP's steadily shrinking time-to-threshold with increasing batch size demonstrates its better large-batch scalability and clear advantage over first- and second-order alternatives. Moreover, our FOP results compare favorably even to recent second-order methods. For instance, SENG (Yang et al. 2020) reaches 75.9% Top-1 on ImageNet-1K in 41 epochs at BS=4096, and SP-NGD (Osawa et al. 2020) only hits 74.8%–75.3% at BS=4096–8192. In contrast, FOP matches or exceeds their results in fewer epochs and achieves the same threshold in 34 epochs at BS=4096 and 40 epochs at BS=8192.

CIFAR10-LT with ResNet32

Finally, to assess optimizer robustness under long-tailed data, we evaluate KFAC and FOP on the CIFAR10-LT and CIFAR100-LT benchmarks (imbalance factors of 100 and 50) using a lightweight ResNet-32, following the training protocol of Zhang et al. (2021). We apply both second-order methods as preconditioners with a fixed damping ratio of $1e-3$ and obtain results over 5 random seeds.

Table 4 reports the Top-1 error rates on the CIFAR-LT datasets under two imbalance factors (50 and 100). We compare our implementations of KFAC and FOP against baseline results from Zhang et al. (2021) and representative results from other recent works (Liu et al. 2019; Kang et al. 2019; Cui et al. 2019). FOP consistently delivers the lowest error, outperforming the baseline by 2.3–3.3% smaller er-

ror rate and KFAC by 1.8–2.0% smaller error rate across all settings. For example, on CIFAR-100-LT with factor 100, it cuts the error from 62.27% (baseline) to 58.97% (3.3% drop), and on CIFAR-10-LT with factor 50, it reduces error from 23.55% to 20.55% (3.0% drop), highlighting its robustness under severe class imbalance. By contrast, KFAC delivers only modest reduction ($\approx 1\%$) on the CIFAR-LT dataset with an imbalance factor of 50, and even underperforms the baseline/other works on those with the factor of 100 (28.59% vs. 28.05% in CIFAR-10-LT, and 61.78% vs. 61.68% in CIFAR-100-LT). These results highlight FOP's superior robustness under severe class imbalance, a benefit we attribute to the Fisher–Orthogonal Projection term that balances curvature estimates across head and tail classes.

Datasets	CIFAR-10-LT		CIFAR-100-LT	
Imbalance Factor	100	50	100	50
Backbones	ResNet-32			
Baselines	28.05	23.55	62.27	56.22
Other works	29.64	25.19	61.68	56.15
KFAC	28.59 ↑	22.29 ↓	61.78 ↑	55.02 ↓
FOP (ours)	**26.65** ↓	**20.55** ↓	**58.97** ↓	**53.67** ↓

Table 4: Top-1 error rates on CIFAR-LT, comparing KFAC and FOP against the implementation baseline (Zhang et al. 2021) and prior results (Liu et al. 2019; Kang et al. 2019; Cui et al. 2019). ↓ indicates that an error rate is lower (better) than both the baseline and other reported results; ↑ indicates that it is higher (worse).

Conclusion

In this work, we introduced Fisher–Orthogonal Projection (FOP), a novel second-order optimizer that combines natural-gradient updates with a geometry-aware variance correction. This design eliminates the need for task-specific tuning when scaling to multiple GPUs in large-batch training. FOP enables seamless scaling to extremely large batch sizes, without requiring any additional tricks or adaptive hyperparameter adjustments, except scaling the learning rate as the batch size. In contrast, standard optimizers such as SGD, AdamW, and KFAC often collapse or demand extensive tuning under such conditions. We validate FOP through extensive experiments on four diverse benchmarks: ResNet-18 on CIFAR-10, T2T-ViT on ImageNet-100, ResNet-50 on ImageNet-1K, and ResNet-32 on long-tailed CIFAR-LT. Across these settings, FOP consistently accelerates convergence and scales more robustly to large batches. Moreover, under severe class imbalance in CIFAR-LT dataset, FOP delivers better generalization, reducing Top-1 error by 2.3–3.3% compared to strong baselines. Together, these results highlight FOP's unique ability to unlock stable, plug-and-play large-batch training across both convolutional and transformer architectures. This makes it especially well-suited for large-scale distributed and resource-constrained environments, paving the way for practical, reliable second-order optimization at scale.

References

Amari, S.; and Nagaoka, H. 2000. Methods of information geometry.

Amari, S.-I. 1998. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276.

Cui, Y.; Jia, M.; Lin, T.-Y.; Song, Y.; and Belongie, S. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9268–9277.

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

DeVries, T.; and Taylor, G. W. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv preprint arXiv:1708.04552*.

Eschenhagen, R.; Immer, A.; Turner, R.; Schneider, F.; and Hennig, P. 2023. Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, 36: 33624–33655.

Ghorbani, B.; Krishnan, S.; and Xiao, Y. 2019. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, 2232–2241. PMLR.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

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

Ishikawa, S.; and Yokota, R. 2024. When Does Second-Order Optimization Speed Up Training? In *The Second Tiny Papers Track at ICLR 2024*.

Kang, B.; Xie, S.; Rohrbach, M.; Yan, Z.; Gordo, A.; Feng, J.; and Kalantidis, Y. 2019. Decoupling representation and classifier for long-tailed recognition. *arXiv preprint arXiv:1910.09217*.

Keskar, N. S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; and Tang, P. T. P. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Li, H.; Xu, Z.; Taylor, G.; Studer, C.; and Goldstein, T. 2018. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31.

Lin, W.; Dangel, F.; Eschenhagen, R.; Neklyudov, K.; Kristiadi, A.; Turner, R. E.; and Makhzani, A. 2024. Structured Inverse-Free Natural Gradient Descent: Memory-Efficient & Numerically-Stable KFAC. In *International Conference on Machine Learning (ICML)*.

Liu, X.; Li, S.; Gao, K.; and Wang, B. 2024. A layer-wise natural gradient optimizer for training deep neural networks. *Advances in Neural Information Processing Systems*, 37: 28460–28489.

Liu, Z.; Miao, Z.; Zhan, X.; Wang, J.; Gong, B.; and Yu, S. X. 2019. Large-scale long-tailed recognition in an open world. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2537–2546.

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Martens, J.; and Grosse, R. 2015. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, 2408–2417. PMLR.

Mattson, P.; Cheng, C.; Coleman, C.; Diamos, G.; ...; Reddi, V. J.; and Zaharia, M. 2019. MLPerf Training Benchmark. arXiv:1910.01500.

McCandlish, S.; Kaplan, J.; Amodei, D.; and Team, O. D. 2018. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*.

Osawa, K.; Ishikawa, S.; Yokota, R.; Li, S.; and Hoefler, T. 2023. ASDL: A Unified Interface for Gradient Preconditioning in PyTorch. *arXiv e-prints*, arXiv–2305.

Osawa, K.; Tsuji, Y.; Ueno, Y.; Naruse, A.; Foo, C.-S.; and Yokota, R. 2020. Scalable and practical natural gradient for large-scale deep learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1): 404–415.

Pascanu, R.; and Bengio, Y. 2013. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*.

Pauloski, J. G.; Huang, L.; Xu, W.; Chard, K.; Foster, I. T.; and Zhang, Z. 2022. Deep neural network training with distributed K-FAC. *IEEE Transactions on Parallel and Distributed Systems*, 33(12): 3616–3627.

Robbins, H.; and Monro, S. 1951. A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Sagun, L.; Bottou, L.; and LeCun, Y. 2016. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*.

Shazeer, N.; and Stern, M. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, 4596–4604. PMLR.

Team, P. 2017. PyTorch Examples. https://github.com/pytorch/examples. Accessed: 2025-08-03.

Yang, M.; Xu, D.; Wen, Z.; Chen, M.; and Xu, P. 2020. Sketchy empirical natural gradient methods for deep learning. *arXiv preprint arXiv:2006.05924*.

Yao, Z.; Gholami, A.; Arfeen, D.; Liaw, R.; Gonzalez, J.; Keutzer, K.; and Mahoney, M. 2018. Large batch size training of neural networks with adversarial training and second-order information. *arXiv preprint arXiv:1810.01021*.

Yuan, L.; Chen, Y.; Wang, T.; Yu, W.; Shi, Y.; Jiang, Z.-H.; Tay, F. E.; Feng, J.; and Yan, S. 2021. Tokens-to-Token ViT: Training Vision Transformers From Scratch on ImageNet. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 558–567.

Zhang, Y.; Wei, X.; Zhou, B.; and Wu, J. 2021. Bag of Tricks for Long-Tailed Visual Recognition with Deep Convolutional Neural Networks. In *AAAI*, 3447–3455.

Supplementary Material

A. Notations and Lemmas

Lemma 1. *Let $g_{diff}, g_{avg} \in \mathbb{R}^n$, and let $F \in \mathbb{R}^{n \times n}$ be a symmetric positive semi-definite matrix (the Fisher information matrix). Define the scalar projection:*

$$s_{FOP} := \frac{g_{diff}^\top F g_{avg}}{g_{avg}^\top F g_{avg} + \epsilon}$$

and the orthogonal component:

$$g_{diff}^\perp := g_{diff} - s_{FOP}\, g_{avg}$$

for some small constant ϵ. Then the Fisher inner product between $g_{diff}^\perp$ and g_{avg} satisfies:

$$\left(g_{diff}^\perp\right)^\top F g_{avg} = g_{diff}^\top F g_{avg} \cdot \left(\frac{\epsilon}{g_{avg}^\top F g_{avg} + \epsilon}\right)$$

In particular:

- *If $\epsilon = 0$, then $g_{diff}^\perp$ is exactly F-orthogonal to g_{avg}.*
- *For $\epsilon > 0$, the deviation from orthogonality is bounded by:*

$$\left|\left(g_{diff}^\perp\right)^\top F g_{avg}\right| \le \epsilon \|F^{1/2} g_{diff}\| \cdot \|F^{-1/2} g_{avg}\|$$

Proof. We start with the definition:

$$g_{\text{diff}}^\perp = g_{\text{diff}} - s_{\text{FOP}}\, g_{\text{avg}}, \quad s_{\text{FOP}} = \frac{g_{\text{diff}}^\top F g_{\text{avg}}}{g_{\text{avg}}^\top F g_{\text{avg}} + \epsilon}$$

Now take the Fisher inner product of $g_{\text{diff}}^\perp$ with g_{avg}:

$$
\begin{aligned}
\left(g_{\text{diff}}^\perp\right)^\top F g_{\text{avg}} &= g_{\text{diff}}^\top F g_{\text{avg}} - s_{\text{FOP}} \cdot g_{\text{avg}}^\top F g_{\text{avg}} \\
&= g_{\text{diff}}^\top F g_{\text{avg}} - \frac{g_{\text{diff}}^\top F g_{\text{avg}}}{g_{\text{avg}}^\top F g_{\text{avg}} + \epsilon} \cdot g_{\text{avg}}^\top F g_{\text{avg}} \\
&= g_{\text{diff}}^\top F g_{\text{avg}} \left(1 - \frac{g_{\text{avg}}^\top F g_{\text{avg}}}{g_{\text{avg}}^\top F g_{\text{avg}} + \epsilon}\right) \\
&= g_{\text{diff}}^\top F g_{\text{avg}} \cdot \frac{\epsilon}{g_{\text{avg}}^\top F g_{\text{avg}} + \epsilon}
\end{aligned}
$$

This proves the first part of the lemma. If $\epsilon = 0$, the expression is zero, yielding exact orthogonality:

$$\left(g_{\text{diff}}^\perp\right)^\top F g_{\text{avg}} = 0$$

For $\epsilon > 0$, we apply the Cauchy-Schwarz inequality under the Fisher inner product:

$$
\begin{aligned}
\left|\left(g_{\text{diff}}^\perp\right)^\top F g_{\text{avg}}\right| &= \left|g_{\text{diff}}^\top F g_{\text{avg}} \cdot \frac{\epsilon}{g_{\text{avg}}^\top F g_{\text{avg}} + \epsilon}\right| \\
&\le \epsilon \cdot \|F^{1/2} g_{\text{diff}}\| \cdot \|F^{-1/2} g_{\text{avg}}\|
\end{aligned}
$$

$\square$

Lemma 2. *Let $F \succ 0$ be a symmetric positive definite matrix, and let $\mu_{\min}$ and $\mu_{\max}$ denote the minimum and maximum eigenvalues of F, respectively. Then for any vector $x \in \mathbb{R}^n$, the following inequality holds:*

$$\mu_{\min}^{1/2} \|F^{-1/2} x\| \le \|x\| \le \mu_{\max}^{1/2} \|F^{-1/2} x\|.$$

Proof. Since $F \succ 0$, it is diagonalizable with positive eigenvalues $\mu_1, \ldots, \mu_n$ and orthonormal eigenvectors. Let $x \in \mathbb{R}^n$. The standard Euclidean norm is:

$$\|x\|^2 = x^\top x,$$

and the Fisher-scaled norm is:

$$\|F^{-1/2} x\|^2 = x^\top F^{-1} x.$$

From spectral theory, the matrix inequality holds:

$$\mu_{\min} I \preceq F \preceq \mu_{\max} I,$$

which implies (after inverting all sides):

$$\mu_{\max}^{-1} I \preceq F^{-1} \preceq \mu_{\min}^{-1} I.$$

Now apply these bounds to the quadratic form $x^\top F^{-1} x$:

$$
\begin{aligned}
x^\top F^{-1} x &\le \mu_{\min}^{-1} x^\top x = \mu_{\min}^{-1} \|x\|^2, \\
x^\top F^{-1} x &\ge \mu_{\max}^{-1} x^\top x = \mu_{\max}^{-1} \|x\|^2.
\end{aligned}
$$

Rewriting in terms of norms:

$$
\begin{aligned}
\|F^{-1/2} x\|^2 \le \mu_{\min}^{-1} \|x\|^2 &\quad \Rightarrow \quad \|x\| \ge \mu_{\min}^{1/2} \|F^{-1/2} x\|, \\
\|F^{-1/2} x\|^2 \ge \mu_{\max}^{-1} \|x\|^2 &\quad \Rightarrow \quad \|x\| \le \mu_{\max}^{1/2} \|F^{-1/2} x\|.
\end{aligned}
$$

$\square$

B. KL-norm analysis

B.1. KL-norm of FOP

In natural gradient methods, such as K-FAC (Martens and Grosse 2015), they define the direction in parameter space which gives the largest change in the objective per unit of change in the model (Amari and Nagaoka 2000), as measured by the Kullback–Leibler (KL) divergence between the current model p_θ and the updated model $p_{\theta+\Delta}$.

By applying a second-order Taylor expansion of the KL divergence around θ, we obtain the following approximation:

$$\text{KL}(p_{\theta+\Delta} \,\|\, p_\theta) \approx \frac{1}{2}\Delta^\top F \Delta = \frac{1}{2}\|F^{1/2}\Delta\|^2 \qquad (23)$$

where F is the Fisher information matrix, and $\|F^{1/2}\Delta\|^2$ is known as the KL-norm of the update step Δ. This KL-norm measures how far the update moves the model in distribution space, making it a more meaningful metric for optimization in probabilistic models.

In our paper, the update rule in the Fisher Orthogonal Projection (FOP) method:

$$\Delta_{\text{FOP}} = -\eta M^{-1}(g + \beta g^\perp) \qquad (24)$$

where:

- $M = F + \lambda I$ is the damped Fisher matrix,

- $\beta \in \mathbb{R}$ controls the contribution from the orthogonal component $g^\perp$,
- η is the learning rate,
- $g^\perp \equiv g^\perp_{\text{diff}}$ and $g \equiv g_{\text{avg}}$
- $g^\perp$ is orthogonal to g in the Fisher metric: $(g^\perp)^\top F g = 0$.

To compute the KL-norm of this FOP update, we evaluate:

$$\|F^{1/2}\Delta_{\text{FOP}}\|^2 = \eta^2 (g + \beta g^\perp)^\top M^{-1} F M^{-1} (g + \beta g^\perp) \tag{25}$$

Let

$$Q = M^{-1} F M^{-1} \tag{26}$$

which is symmetric and positive semi-definite. Then the KL-norm becomes:

$$\begin{aligned}
\left\|F^{1/2}\Delta_{\text{FOP}}\right\|^2 &= \eta^2 (g + \beta g^\perp)^\top Q (g + \beta g^\perp) \\
&= \eta^2 \left[g^\top Q g + 2\beta g^\top Q g^\perp + \beta^2 (g^\perp)^\top Q g^\perp \right]
\end{aligned} \tag{27}$$

where $g^\top Q g$ is the KL norm of KFAC, $2\beta\, g^\top Q g^\perp$ is a cross term and $\beta^2 (g^\perp)^\top Q g^\perp$ is contributing the orthogonal projection. Each term captures a different component of the curvature-informed update.

The first term $g^\top Q g$ was proved to be situated in the trust region (Martens and Grosse 2015), so we now focus on bounding the cross term $g^\top Q g^\perp$, and the orthogonal term $\beta^2 (g^\perp)^\top Q g^\perp$. Bounding these terms is essential to ensure that the KL-norm remains predictable under varying damping parameters.

Bounding $2\beta\, g^\top Q g^\perp$
Substituting Eq. 26 into the cross term gives:

$$g^\top Q g^\perp = g^\top M^{-1} g^\perp - \lambda g^\top M^{-2} g^\perp \tag{28}$$

and we can bound this:

$$|g^\top Q g^\perp| = \left| g^\top M^{-1} g^\perp - \lambda g^\top M^{-2} g^\perp \right| \tag{29}$$

and with triangle inequality ($|a - b| \le |a| + |b|$):

$$|g^\top Q g^\perp| \le |g^\top M^{-1} g^\perp| + \lambda |g^\top M^{-2} g^\perp| \tag{30}$$

Applying the Cauchy–Schwarz inequality to each term:
For the first term:

$$|g^\top M^{-1} g^\perp| \le \|M^{-1/2} g\| \cdot \|M^{-1/2} g^\perp\| \tag{31}$$

For the second term:

$$|g^\top M^{-2} g^\perp| \le \|M^{-1} g\| \cdot \|M^{-1} g^\perp\| \tag{32}$$

Since $M^{-1} \preceq \lambda^{-1} I$, it follows that:

$$\|M^{-1/2} g\| \le \lambda^{-1/2} \|g\|, \quad \|M^{-1} g\| \le \lambda^{-1} \|g\| \tag{33}$$

Putting Eq. 33 into Eq. 31 and Eq. 32:

$$\|M^{-1/2} g\| \cdot \|M^{-1/2} g^\perp\| \le \lambda^{-1} \|g\| \cdot \|g^\perp\| \tag{34}$$

$$\lambda \cdot \|M^{-1} g\| \cdot \|M^{-1} g^\perp\| \le \lambda \cdot \lambda^{-1} \cdot \lambda^{-1} \|g\| \cdot \|g^\perp\|$$
$$= \lambda^{-1} \|g\| \cdot \|g^\perp\| \tag{35}$$

Putting these back to Eq. 30, we get:

$$|g^\top Q g^\perp| \le \lambda^{-1} \|g\| \cdot \|g^\perp\| + \lambda^{-1} \|g\| \cdot \|g^\perp\| = \frac{2}{\lambda} \|g\| \cdot \|g^\perp\| \tag{36}$$

To express this in terms of the $F^{-1/2}$-norm, recall *Lemma 2*:

$$\|g\| \le \mu_{\max}^{1/2} \|F^{-1/2} g\|, \quad \|g^\perp\| \le \mu_{\max}^{1/2} \|F^{-1/2} g^\perp\|$$

Therefore:

$$|g^\top Q g^\perp| \le \frac{2\mu_{\max}}{\lambda} \|F^{-1/2} g\| \cdot \|F^{-1/2} g^\perp\| \tag{37}$$

Bounding $\beta^2 (g^\perp)^\top Q g^\perp$
From Eq. 26:

$$Q = (F + \lambda I)^{-1} F (F + \lambda I)^{-1} \tag{38}$$

In the eigenbasis of F (symmetric and positive semi-definite), where $F = \text{diag}(\Lambda_1, \ldots, \Lambda_n)$ and Λ_i is the eigenvalue, we can express Q as a diagonal matrix with entries:

$$Q_i = \frac{\Lambda_i}{(\Lambda_i + \lambda)^2} \tag{39}$$

Let $g^\perp = \sum_i h_i u_i$, where $\{u_i\}$ are the eigenvectors of F, and $h_i = u_i^\top g^\perp$. Then:

$$(g^\perp)^\top Q g^\perp = \sum_i \frac{\Lambda_i}{(\Lambda_i + \lambda)^2} h_i^2 \tag{40}$$

This achieves its maximum at $\Lambda_i = \lambda$, therefore:

$$(g^\perp)^\top Q g^\perp \le \frac{1}{4\lambda} \sum_i h_i^2 = \frac{1}{4\lambda} \|g^\perp\|^2 \tag{41}$$

Recall *Lemma 2*, we have:

$$\|g^\perp\| \le \mu_{\max}^{1/2} \|F^{-1/2} g^\perp\|$$

Substitute into the bound:

$$\beta^2 (g^\perp)^\top Q g^\perp \le \frac{\beta^2}{4\lambda} \mu_{\max} \|F^{-1/2} g^\perp\|^2 \tag{42}$$

Finally, Eq. 27 becomes:

$$\begin{aligned}
\|F^{1/2}\Delta_{\text{FOP}}\|^2 \le \eta^2 \bigg[&\underbrace{g^\top Q g}_{\|F^{1/2}\Delta_{\text{KFAC}}\|^2} \\
&+ 2\beta \frac{\mu_{\max}}{\lambda} \|F^{-1/2} g\| \cdot \|F^{-1/2} g_\perp\| \\
&+ \beta^2 \frac{\mu_{\max}}{4\lambda} \|F^{-1/2} g_\perp\|^2 \bigg]
\end{aligned} \tag{43}$$

The KL-norm of an FOP update eventually decomposes into three terms: the K-FAC core term, which shrinks quadratically as λ^{-2}, and two FOP-specific terms, a cross term and an orthogonal term, which shrink linearly as

λ^{-1}. Crucially, the FOP-specific terms are also scaled by $\|F^{-1/2}g^\perp\|$, which is typically smaller than $\|F^{-1/2}g\|$.

The analysis deliberately combines an exact expression for the core term with a worst-case bound for the orthogonal term to ensure safety even when the spectrum of $g^\perp$ is unknown. In the large-damping training that K-FAC relies on for stability at big batch sizes, this bound guarantees that the FOP-specific terms remain at least one order of λ smaller than the core term.

This allows FOP to reduce λ, preserving more curvature information while still respecting the KL trust-region. As a result, FOP achieves faster convergence in large-batch training without compromising stability.

C. Experiments

C.1. Setup of CIFAR-10

The training of ResNet-18 (He et al. 2016) on the CIFAR-10 (Krizhevsky, Hinton et al. 2009) dataset serves as a fundamental experiment in the field of image classification. In this subsection, we compare the proposed FOP optimizer with several established baselines, including SGD with momentum (referred to as SGD (Robbins and Monro 1951)), AdamW (Loshchilov and Hutter 2017), and KFAC. We follow standard experimental settings and apply commonly used data augmentation techniques consisting of random cropping and horizontal flipping (DeVries and Taylor 2017). All models are trained for 100 epochs. For SGD, KFAC, and FOP, the initial learning rate α_0 for batch size of 1024 is selected via grid search over the set $\alpha \in \{10^{-2}, 5 \times 10^{-2}, 10^{-1}, 5 \times 10^{-1}, 1\}$. For AdamW, the grid is set to $\alpha \in \{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}\}$. The update intervals for the curvature matrix and its inverse in both KFAC and FOP are synchronised, and we evaluate two base intervals: $\{10, 100, 200\}$. Each experiment is repeated with 5 different random seeds ($\{0, 1, 2, 3, 4\}$) to account for variability. We use ReduceLROnPlateau as the learning rate scheduler, configured with a patience of 5 epochs, a decay factor of 0.1, a relative threshold of 0.0001, and threshold_mode set to rel.

C.2. Setup of ImageNet-100 experiment

The implementation of T2T-ViT (Yuan et al. 2021) follows the original paper. We use the linear warmup strategy (Goyal et al. 2017) in the first 5 epochs for AdamW, KFAC and FOP. The initial learning rate α_0 is selected via grid search over $\alpha \in \{10^{-5}, 10^{-4}, \dots, 10^{-1}\}$ for a batch size of 512. .The update intervals for the curvature matrix and inverse matrix correlating with KFAC and FOP are set to be $\{500, 1000\}$ for a batch size of 512. All models are trained for 100 epochs with three random seeding ($\{0, 1, 2\}$). AdamW, KFAC, and FOP use the cosine learning rate updating strategy and are set to be

$$\alpha_t = 0.001 + 0.5 \times (\alpha_0 - 0.001)$$
$$\times \left(1 + \cos\left(\frac{2 \times 0.47 \times \pi \times t}{\text{max\_epoch}}\right)\right)$$
$$(44)$$

C.3. Setup of ImageNet-1K experiment

We implement ResNet50 following the official PyTorch example (Team 2017), based on the architecture proposed by He et al. (He et al. 2016). For all optimizers (SGD, Adam, KFAC, and FOP), we apply a linear warm-up strategy (Goyal et al. 2017) during the first 5 epochs. For the KFAC and FOP optimizers, the update intervals for the curvature matrix and its inverse are set to $\{800, 1600, 2400\}$ when using a batch size of 1024. The total number of training epochs differs by optimizer: we train SGD and Adam for 80 epochs, while KFAC and FOP are trained for 50 epochs. SGD uses the cosine annealing learning rate schedule as described in Eq. 44. In contrast, KFAC and FOP adopt an exponential decay learning rate schedule given by:

$$\alpha_t = \alpha_0 \times \left(1 - \frac{t}{\text{max\_epoch}}\right)^E$$

where t is the current epoch, and E is the decay exponent, with $E \in \{4, 5, 6\}$. The initial learning rate α_0 is selected via grid search over $\alpha \in \{10^{-5}, 10^{-4}, \dots, 10^{-1}\}$ for a batch size of 1024.

C.4. Additional experimental details and results

C.4.1 Damping ratio vs acc   Figure 6 presents results on CIFAR-10 with ResNet-18. Across all batch sizes (2048–50000), the proposed FOP optimiser consistently surpasses the 91 % test-accuracy threshold (dotted line) when the damping ratio $\lambda \in [10^{-5}, 10^{-3}]$. In contrast, KFAC fails to meet the cutoff whenever the damping is too aggressive ($\lambda \geq 10^{-2}$) and becomes unstable at $\lambda = 10^{-5}$. While both methods perform best around $\lambda = 10^{-3}$, FOP's flatter curve indicates a much larger tolerance to hyper-parameter misspecification.

This pattern extends to transformer-based models as well. As shown in Figure 7, for T2T-ViT trained on ImageNet-100, FOP reliably exceeds the 80.9 % bar for all batch sizes and for all damping values except the extreme low end. In contrast, KFAC only reaches the target within the narrow range $\lambda \in [10^{-4}, 10^{-3}]$. Notably, FOP's accuracy varies by less than one percentage point across the entire sweep from $\lambda = 10^{-5}$ to 10^{-1} at smaller batch sizes, highlighting its robustness even on architectures that include attention layers. A similar trend appears on the large-scale ImageNet-1K task with ResNet-50, as illustrated in Figure 8. Here, FOP again outperforms or matches KFAC across the board and remains above the 75.9 % accuracy threshold throughout the full sweep and across all batch sizes (2048–8196). KFAC's performance, however, drops sharply for $\lambda \geq 10^{-2}$ and exhibits visible instability at the smallest damping, echoing the behaviour observed on CIFAR-10. Collectively, Figures 6–8 indicate that FOP provides a substantially broader and more stable operating range for the damping ratio compared to KFAC, across both convolutional and transformer-based architectures, and over a wide range of data scales. This increased robustness reduces the sensitivity to hyper-parameter selection and supports the viability of FOP as a reliable second-order optimisation method for large-batch training in diverse tasks.
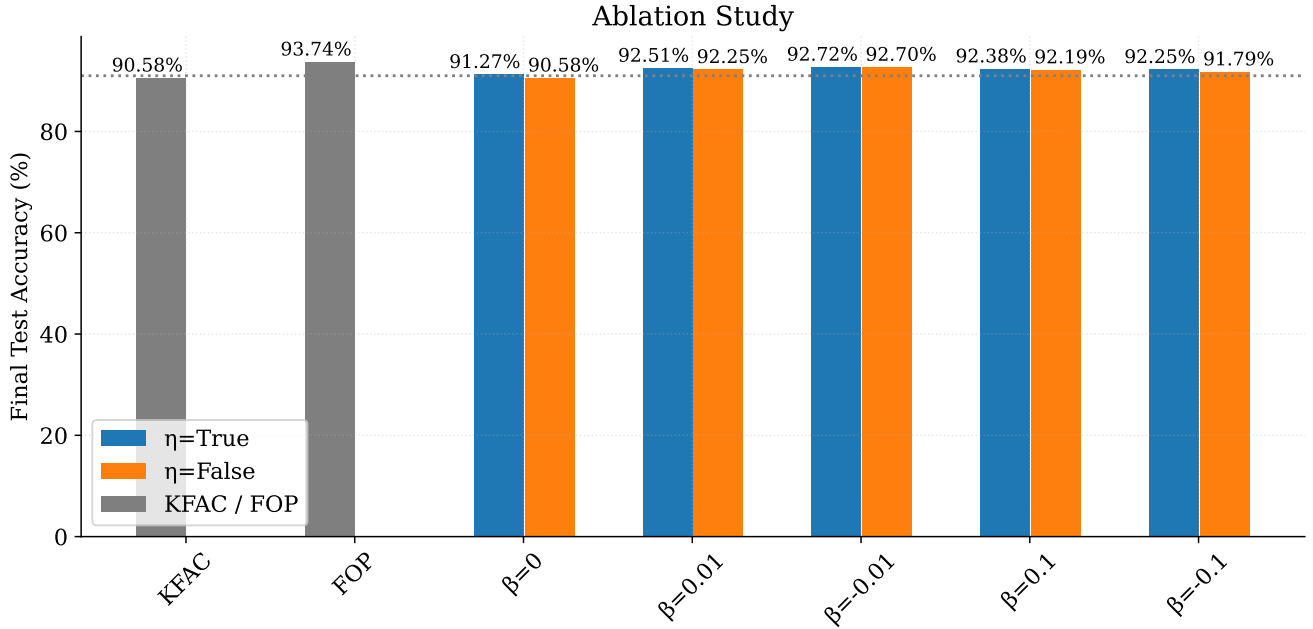
Figure 5: **Ablation of the scaling parameter η on CIFAR-10.** Bars are grouped by optimiser KFAC (left cluster) and FOP (right cluster), and coloured by the value of the scaling term $\eta \in \{-0.1, -0.01, 0, 0.01, 0.1\}$. Numbers above each bar give the Top 3 test accuracy averaged over three random seeds.
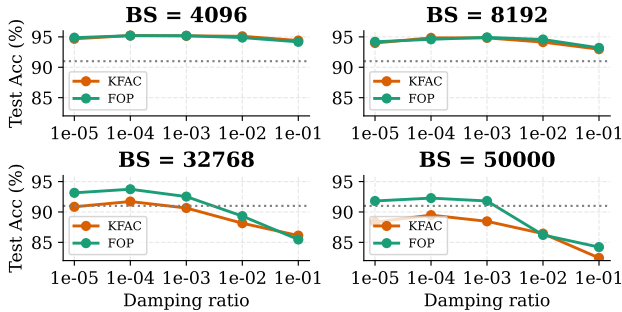

Figure 6: Best test accuracy vs. damping ratio for ResNet-18 on CIFAR-10, grouped by batch size. The dotted line represents the threshold of 91%.
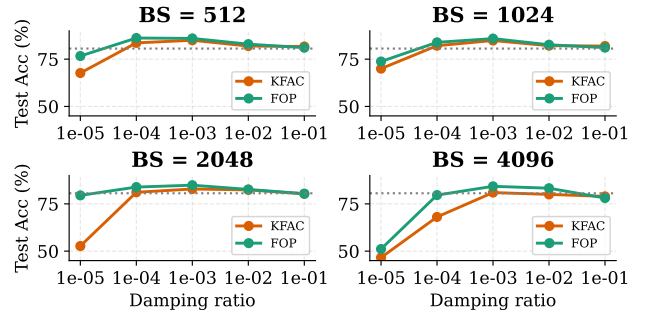

Figure 7: Best test accuracy vs. damping ratio for T2T-ViT on ImageNet-100, grouped by batch size. The dotted line represents the threshold of 80.9%.

C.4.2 GPU memory Tables 5–7 report the peak GPU memory usage for CIFAR-10/ResNet-18, ImageNet-100/T2T-ViT, and ImageNet-1K/ResNet-50, respectively, along with the number of devices used in each run. On CIFAR-10 (Table 5), both SGD and AdamW exhibit the lowest memory footprint, with usage scaling nearly linearly with batch size. KFAC incurs a moderate overhead, while FOP demands the highest memory—up to 145 GB for a batch of 16 384 on a single GPU, as it needs to keep two gradients g_1 and g_2 under the same device. This vanishes when there are more than 1 device. For ImageNet-100 with T2T-ViT (Table 6), all three methods operate near full device capacity ($\approx$150–170 GB) at the smallest batch size,

with FOP consistently exceeding the others by 5–20 GB. On ImageNet-1K with ResNet-50 (Table 7), memory per GPU remains constant within each optimiser regardless of batch size, since larger batches are distributed across more devices. Here, SGD requires ≈ 88 GB per GPU, KFAC ≈ 98 GB, and FOP again matches KFAC closely except for the smallest batch where it peaks at 153 GB.
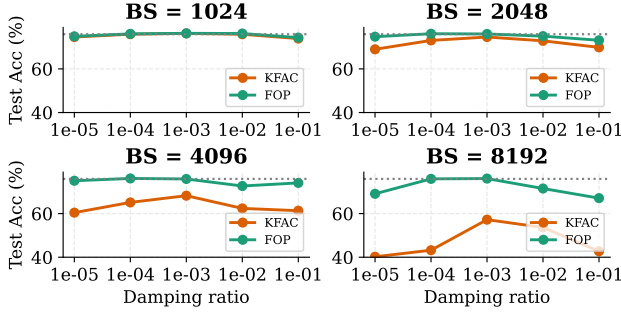
Figure 8: Best test accuracy vs. damping ratio for ResNet-50 on ImageNet-1K, grouped by batch size. The dotted line represents the threshold of 75.9%.

Optimizer	# of GPU	Max Mem/GPU (GB)
Batch Size: 2048		
SGD	1	9.25
AdamW	1	9.30
KFAC	1	14.51
FOP	1	19.21
Batch Size: 4096		
SGD	1	18.40
AdamW	1	18.44
KFAC	1	28.03
FOP	1	37.20
Batch Size: 8192		
SGD	1	36.70
AdamW	1	36.74
KFAC	1	55.08
FOP	1	73.18
Batch Size: 16384		
SGD	1	73.29
AdamW	1	73.33
KFAC	1	109.18
FOP	1	145.16
Batch Size: 32768		
SGD	2	73.29
AdamW	2	73.38
KFAC	2	108.31
FOP	2	108.44

Table 5: Max GPU memory usage and number of GPUs for CIFAR-10 training with ResNet-18.

Optimizer	# of GPU	Max Memory/GPU (GB)
Batch Size: 512		
AdamW	1	145.41
KFAC	1	153.28
FOP	1	172.51
Batch Size: 1024		
AdamW	1	151.10
KFAC	1	158.85
FOP	1	163.18
Batch Size: 2048		
AdamW	1	148.54
KFAC	1	156.86
FOP	1	159.20
Batch Size: 4096		
AdamW	1	142.95
KFAC	1	151.20
FOP	1	153.66

Table 6: Max GPU memory usage across optimizers and number of GPUs used per run for training ImageNet-100 with T2T-ViT.

Optimizer	# of GPU	Max Memory/GPU (GB)
Batch Size: 1024		
SGD	1	87.89
KFAC	1	110.31
FOP	1	153.31
Batch Size: 2048		
SGD	2	87.99
KFAC	2	98.04
FOP	2	98.24
Batch Size: 4096		
SGD	4	87.99
KFAC	4	98.04
FOP	4	98.24
Batch Size: 8192		
SGD	8	87.99
KFAC	8	97.89
FOP	8	98.19

Table 7: Max GPU memory usage and number of GPUs used per run for training ImageNet-1K with ResNet-50.
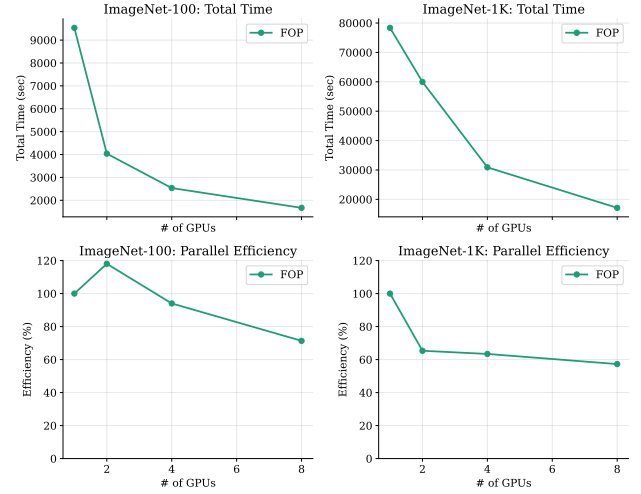
C.4.3 Scaling efficiency

Figure 9: Strong-scaling results for FOP on ImageNet-1K. Top: wall-clock training time versus number of GPUs for ImageNet-100 (left) and ImageNet-1K (right). Bottom: total GPU-time (#GPUs × wall-clock) versus number of GPUs for the same two datasets. Dashed lines indicate ideal linear scaling.

D. Ablation Study of FOP

D.1. Effect of β

An ablation study on CIFAR-10, shown in Figure 5, evaluates the effect of varying the momentum parameter β in the FOP optimiser. Using ResNet-18 trained for 100 epochs (averaged over three seeds), the study compares different configurations of FOP against KFAC, which fixes $(\eta, \beta) = (1, 0)$. FOP decouples the learning rate scale η from the momentum factor β, allowing both to be fixed or adaptively updated. When η is not adaptive (set to 1), introducing β consistently improves accuracy over KFAC. Further gains are observed when η is adaptive, highlighting the benefit of dynamic learning rate scaling. Overall, the results demonstrate that FOP's flexibility in tuning or adapting (η, β) leads to improved accuracy and robustness compared to fixed-setting baselines.

E. Comparison between Nvidia and AMD cards

E.1. System Configuration

AMD System (Training Runs)

- **Node Type:** Lenovo ThinkSystem SR685a V3
- **CPUs:** 2× AMD EPYC 9534 (64 cores each, 128 total cores, no SMT)
- **Max Frequency:** 3.72 GHz, Frequency Boost enabled
- **Instruction Support:** AVX-512, AVX512-BF16, AVX512-VNNI, AVX512-VBMI
- **Cache:** L3 cache total 512 MiB (16×32 MiB)
- **GPUs:** 8× AMD MI300X
- **Software Stack:** PyTorch 2.6.0 + ROCm 6.2.4
- **ROCm Driver Version:** 6.2.1

NVIDIA System (Benchmarking)

- **Node Type:** Custom system
- **CPUs:** 2× Intel Xeon Platinum 8468 (48 cores each, 96 total cores)
- **Max Frequency:** 3.8 GHz
- **Instruction Support:** AVX-512, AVX-VNNI, AMX (Tile, INT8, BF16)
- **GPUs:** 8× NVIDIA H100 (80 GB HBM3)
- **Software Stack:** PyTorch 2.8.0.dev20250413+cu126, CUDA 12.9
- **Driver Version:** 575.57.08

E.2. Benchmarking Results on ImageNet-1k with ResNet-50

To evaluate both the scalability and efficiency of different optimization methods across hardware platforms, we present four subplots in Figure 10. Subplot (a) shows the *total GPU time per epoch*, which accounts for both the per-step time and the number of GPUs used, reflecting the total computational cost. Subplot (b) illustrates the *raw wall-clock time per epoch* when training with 8 GPUs.

In subplot (c), we measure the **strong scaling parallel efficiency**, defined as

$$\text{Efficiency}_{\#\text{GPUs}} = \frac{T_1}{n \cdot T_n} \times 100\%,$$

where T_1 is the training time using the smallest GPU count (typically one GPU), T_n is the time when using n GPUs, and n is the number of GPUs. This metric indicates how effectively the training process scales with added hardware resources.

Subplot (d) evaluates the **efficiency with respect to batch size at fixed GPU count** (8 GPUs), defined as

$$\text{Efficiency}_{\text{batch}} = \frac{T_{b_0} \cdot b_0}{T_b \cdot b} \times 100\%,$$

where T_{b_0} is the epoch time at a reference batch size b_0, and T_b is the epoch time for a new batch size b. This quantifies how well larger batch sizes maintain per-sample throughput on the same hardware.

Together, these plots provide insights into both inter-GPU scaling behavior and intra-GPU efficiency across diverse training methods and compute platforms.
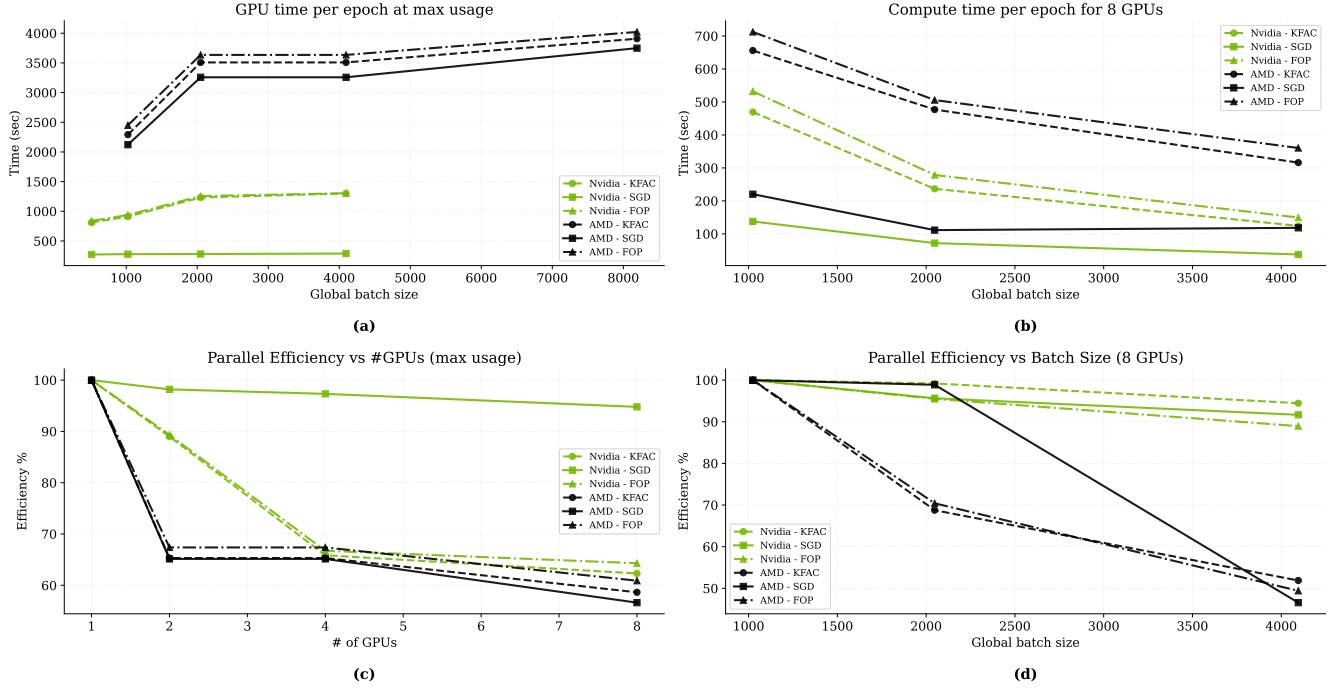
Figure 10: Training time and parallel efficiency comparisons across different methods (SGD, KFAC, FOP) and hardware (NVIDIA H100 and AMD MI300X). (a) Total GPU time per epoch scaled by the number of GPUs used, representing overall computational effort. (b) Raw wall-clock time per epoch using 8 GPUs. (c) **Parallel efficiency vs number of GPUs**, calculated as Efficiency $= \frac{T_1}{nT_n}$, where T_1 is the time with the smallest GPU count and T_n the time using n GPUs. (d) **Efficiency vs global batch size on 8 GPUs**, defined as Efficiency $= \frac{T_{b_0} \cdot b_0}{T_b \cdot b}$, measuring how well increased batch sizes are utilized on fixed hardware.