# Beyond the Mean: Fisher-Orthogonal Projection for Natural Gradient Descent in Large Batch Training

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

[1]Oxford e-Research Centre, Department of Engineering Science, University of Oxford
yishun.lu@eng.ox.ac.uk, wes.armour@oerc.ox.ac.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.

**Code** — https://github.com/yishunlu-222/fop.git
**Extended version** — https://arxiv.org/abs/2508.13898

## 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; You et al. 2019) 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; Shi et al. 2023) improve throughput but still rely on heavily damped mean gradients. Although these methods reduce hardware requirements, 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 $\times 7.5$ 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 $\theta_i$ is the parameter vector at iteration $i$, $\eta$ is the learning rate, and $\nabla_\theta \mathcal{L}(\theta_{i-1})$ is the gradient of the loss function $\mathcal{L}$ evaluated at $\theta_{i-1}$. 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).

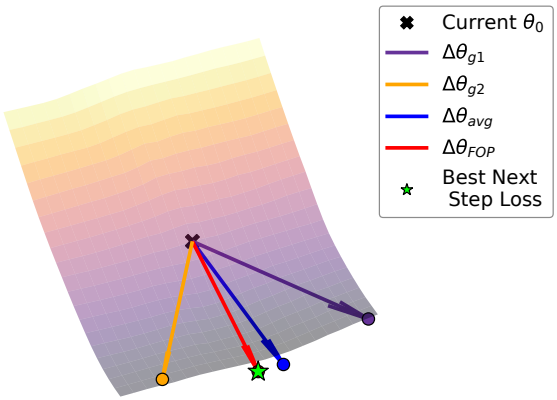


Figure 1: 3D loss landscape of training ResNet-18 with CIFAR-10 for batch size of 1024. This visualization relies on the method suggested in (Li et al. 2018). Arrows represent the direction of the steps of different gradients. The green star is the smallest loss after updating the model based on different 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 $\theta$, 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 $g_{\text{diff}}$ with respect to $g_{\text{avg}}$ 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 Supplementary material in the extended version), 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 $\beta$ 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 $\beta$

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 $\beta$ that balances the direction $g_{\text{diff}}^\perp$ relative to the average gradient. We derive this optimal $\beta$ (denoted $\beta^*$) 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 $L_{\text{tot}}$ around $\theta$, 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 $\beta$, we define a surrogate objective $J_{\text{tot}}(\beta)$ by dropping constants and factors of $\eta$:

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

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:

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

where $\|g_{\text{avg}}\|^2_{F^{-1}} = g_{\text{avg}}^\top F^{-1} g_{\text{avg}}$ is absorbed into the constant term.
To find the optimal $\beta^*$ , we differentiate (14) with respect to $\beta$ 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 $\beta^*$ 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 $\eta_\ell^*$

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 $\eta_\ell^*$. Rather than using a single global learning rate for all parameters, $\eta_\ell^*$ is chosen to (locally) minimize a quadratic approximation of the loss function along the natural-gradient direction for each layer $\ell$. Instead of differentiating about $\beta$ in Eq. 10, we minimize this one-dimensional quadratic model in $\eta$. After we set the derivative with respect to $\eta$ 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, $\eta_\ell^*$ 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, $\eta_\ell^*$ 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 $\eta_0$ 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:

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

where $F$ is the Fisher information matrix and $\Delta$ 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:

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

where $g$ is the average micro-batch gradient $g_{\text{avg}}$, $g^{\perp}$ 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:

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

- 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 $\Lambda_i$ that carries weight in $g$, the spectral factor $\frac{\Lambda_i}{(\Lambda_i+\lambda)^2}$ is proportional to $\Lambda_i/\lambda^2$. Therefore, the base term scales as $g^{\top}Qg \propto \lambda^{-2}$. As shown in Supplementary material in the extended version, the two additional terms from FOP decay even more gently. Specifically:

$$\begin{aligned}\|F^{1/2}\Delta_{\mathrm{FOP}}\|^2 \le \eta^2\Bigg[ &\underbrace{g^{\top}Qg}_{\|F^{1/2}\Delta_{\mathrm{KFAC}}\|^2} \\ &+ 2\beta\frac{\mu_{\max}}{\lambda}\|F^{-1/2}g\|\cdot\|F^{-1/2}g_{\perp}\| \\ &+ \beta^2\frac{\mu_{\max}}{4\lambda}\|F^{-1/2}g_{\perp}\|^2\Bigg]\end{aligned} \tag{22}$$

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 $\lambda$ 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 $\lambda$.

## 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 $\ell_i$, 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 $\tilde{g}$

```
for all p_j ∈ P in parallel do
    g_j ← ∇_θ L(M; D_j) {local back-prop}
    if curvature update step then
        for all layers ℓ_i in M do
            p_k ← specialist s.t. ℓ_i ∈ S_k
            send local curvature factors of ℓ_i to p_k
                                        /* async, non-blocking */
            if j = k then
                update and invert F_i → F_i^{-1}
            end if
        end for
    end if
    if p_j ∈ G_pri then
        global AllReduce to compute g_1
    else
        global AllReduce to compute g_2
    end if
    for all layers ℓ_i in M do
        p_k ← specialist for ℓ_i
        if j = k then
            g̃_i ← FOP(g_{1,i}, g_{2,i}, F_i^{-1})
            broadcast g̃_i to all processes
        end if
    end for
end for
assemble g̃ ← [g̃_1, ..., g̃_L] {L = #layers} =0
```

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

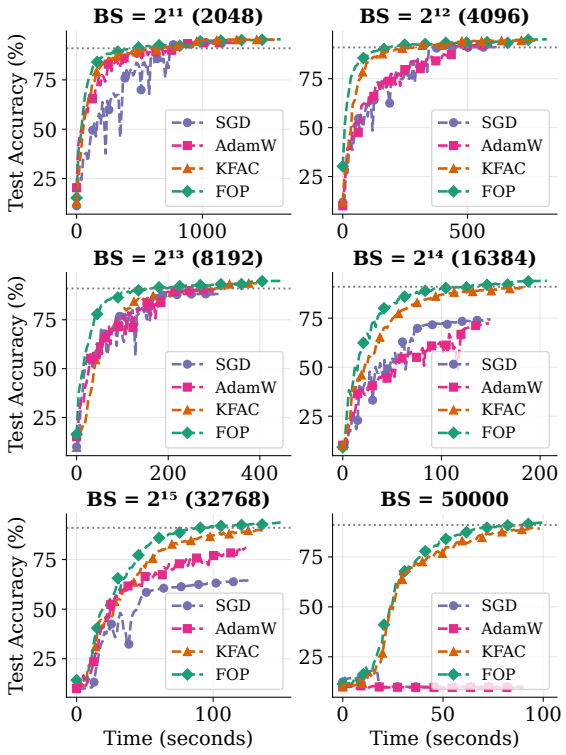


Figure 2: Test accuracy vs. wall-clock time (in seconds) for ResNet-18 on CIFAR-10, grouped by batch size. The dotted line represents the threshold of 91%.

splitting large batches. First, we assign each GPU as a specialist, responsible for updating the curvature (Fisher information) 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 $\lambda$. 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 Supplementary material in the extended version.

### 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 $\times 1.56$ faster than SGD (58/743.3s) and $\times 1.26$ faster than KFAC (37/588.7s). As we scale to BS=4096 and 8192, FOP's speedup over SGD grows to $\times 1.69$ and $\times 2.91$ , respectively, and reaches $\times 3.78$ at BS=16384. Crucially, FOP is the only method to arrive 91% at BS=32768 and 50000, doing so in 60/90.6s ($\times 5.05$) and 82/84.3s ($\times 5.43$). 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

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

ImageNet-100 with running 3 different random seeds(Yuan et al. 2021; Lin et al. 2024). Following Lin et al. (2024), we 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.

| Optimizer | Batch Size (GPU): Epochs / Time (min) | | | |
|---|---|---|---|---|
| | 512 (1) | 1024 (2) | 2048 (4) | 4096 (8) |
| AdamW | 97 / 291.6 | – | – | – |
| KFAC | 42 / 138.6<br>[×2.10] | 49 / 76.7<br>[×3.80] | 57 / 46.0<br>[×6.34] | 87 / 45.3<br>[×6.45] |
| FOP (ours) | 44 / 158.9<br>[×1.84] | 41 / 67.3<br>[×4.33] | 48 / 42.3<br>[×6.90] | 49 / 27.8<br>[×10.48] |

Table 2: Epoch and training time (in minutes) 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.

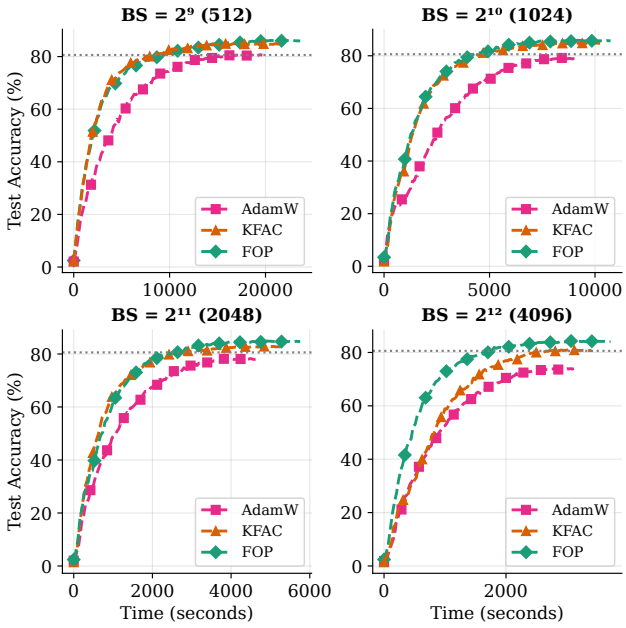


Figure 3: Test accuracy vs. wall-clock time (in seconds) for T2T-ViT on ImageNet-100, grouped by batch size. The dotted line represents the threshold of 80.6%.

## ImageNet-1K with ResNet50

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, Shampoo (Gupta, Koren, and Singer 2018), and LAMB (You et al. 2019) for 80 epochs with a cosine learning-rate schedule. We train both KFAC and FOP for 50 epochs using an exponential schedule on their learning rates, because in the SGD, Shampoo, and LAMB can't reach the threshold of 75.9% within 50 epochs.

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 of K-FAC's 35/1336.5 min (×1.88). Beyond this scale, only FOP succeeds to hit the threshold in 33/999.5 min at BS=2048 (×2.51), 34/514.9 min at BS=4096 (×4.88), and 40/335.1 min at BS=8192 (×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

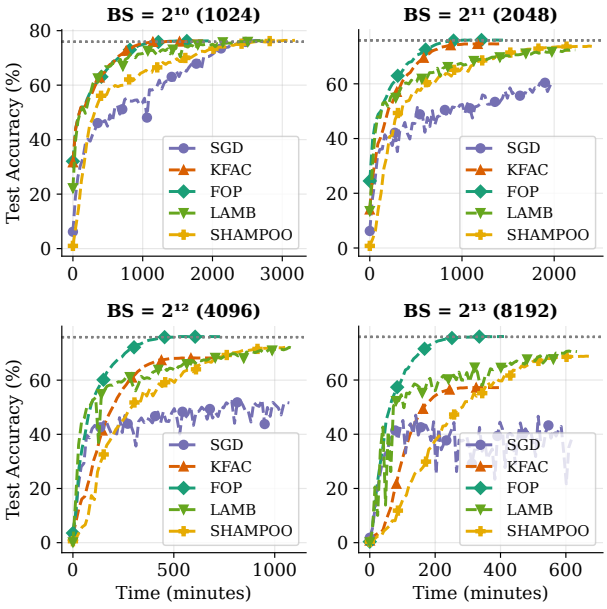


Figure 4: Test accuracy vs. wall-clock time (in minutes) for ResNet-50 on ImageNet-1K, grouped by batch size. The dotted line represents the threshold of 75.9%.

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 error 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.

| **Optimizer** | **Batch Size (GPU): Epochs / Time (min)** | | | |
|---|---|---|---|---|
| | **1024 (1)** | **2048 (2)** | **4096 (4)** | **8192 (8)** |
| SGD | 71 / 2511.1 | – / – | – / – | – / – |
| Shampoo | 65 / 2523.1 | – / – | – / – | – / – |
| LAMB | 67 / 2492.7 | – / – | – / – | – / – |
| KFAC | 35 / 1336.5 [(×1.88)] | – / – | – / – | – / – |
| FOP (ours) | 32 / 1306.1 [(×1.92)] | 33 / 999.5 [(×2.51)] | 34 / 514.9 [(×4.88)] | 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.

| **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.

## Acknowledgments

This work was supported by UK Research and Innovation (UKRI) EP/T022205/1. The authors would also like to acknowledge support from the JADE 2.5 supercomputing service. The authors would also like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility (Richards 2015). The authors also acknowledge the use of resources provided by the Isambard-AI National AI Research Resource (AIRR). Isambard-AI is operated by the University of Bristol and is funded by the UK Government's Department for Science, Innovation and Technology (DSIT) via UK Research and Innovation; and the Science and Technology Facilities Council [ST/AIRR/I-A-I/1023].

## References

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.

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*.

Gupta, V.; Koren, T.; and Singer, Y. 2018. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, 1842–1850. PMLR.

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.

Richards, A. 2015. University of Oxford Advanced Research Computing.

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.

Shi, H.-J. M.; Lee, T.-H.; Iwasaki, S.; Gallego-Posada, J.; Li, Z.; Rangadurai, K.; Mudigere, D.; and Rabbat, M. 2023. A

distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*.

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*.

You, Y.; Li, J.; Reddi, S.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; and Hsieh, C.-J. 2019. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*.

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.