Block-Diagonal K-FAC: A Trade-off Between Curvature Information and Resource Efficiency

Mingzhe Yu YU@HPCS.CS.TSUKUBA.AC.JP
Graduate School of Science and Technology, University of Tsukuba, Tsukuba, Japan

Osamu Tatebe TATEBE@CS.TSUKUBA.AC.JP
Center for Computational Sciences, University of Tsukuba, Tsukuba, Japan

Abstract

We introduce Block-Diagonal K-FAC (BD-KFAC), a second-order optimizer that preserves block-diagonal structure in the Kronecker factors of K-FAC to balance curvature fidelity and resource efficiency. Concretely, we partition activations and pre-activation gradients into per-layer blocks and perform eigendecomposition per block, while precomputing damped block-wise inverses to amortize per-step costs and reduce communication in distributed training. On CIFAR-100 across both CNN and ViT architectures, BD-KFAC achieves faster wall-clock convergence than baselines and uses less memory and computation resource than K-FAC under a unified training setup. Overall, BD-KFAC offers a practical middle ground between full-matrix and purely diagonal second-order methods.

Keywords: K-FAC, second-order optimization, deep learning, block-diagonal approximation

1. Introduction

Second-order optimization methods have attracted growing interest due to their strong convergence and reduced reliance on extensive hyperparameter tuning. However, the cost of forming, storing, and inverting curvature matrices (Hessian/FIM) remains prohibitive at scale, which has motivated approximations that retain useful curvature while controlling computation and memory.

A key observation from existing research [4, 5, 11, 16] is that both the Hessian and the FIM often exhibit near-diagonal structure, meaning that much of the useful curvature information is concentrated near the diagonal. This structural property has motivated a series of algorithms that adopt diagonal or block-diagonal approximations of the preconditioner.

K-FAC [5, 11] stands out as a widely used method that leverages a block-diagonal approximation of the FIM by factoring it into two smaller covariance matrices associated with the inputs and outputs of each layer. K-FAC has demonstrated strong empirical performance across a range of deep learning tasks, but its cost of maintaining and inverting dense covariance factors can still be significant. More recently, AdaFisher [4] highlighted that the K-FAC factors themselves often exhibit diagonal dominance. Building on this insight, AdaFisher proposed a purely diagonal variant that eliminates the need for expensive matrix inversions, offering a lightweight alternative to full K-FAC.

We propose **Block-Diagonal K-FAC (BD-KFAC)**, which retains block-diagonal structure in the K-FAC factors A and G instead of collapsing them to the diagonal. This design aims to strike a balance between curvature fidelity and resource efficiency. Experiments on CIFAR-100 show that

BD-KFAC converges faster than diagonal baselines while using substantially fewer resources than full K-FAC.

2. Related Work

As discussed in the previous section, diagonal or block-diagonal approximations are among the most common techniques in second-order optimization. For block-diagonal approximations, arranging blocks at the granularity of layers is the most natural choice, as exemplified by classical methods such as K-FAC [5, 11] and Shampoo [1]. Some approaches, such as AdaBlock [18] and CASPR [3], attempt to capture inter-layer information in order to obtain a closer approximation to the full-matrix preconditioner. Yen et al. [17] instead focus on further approximating the intra-layer preconditioner, leveraging low-rank matrices and shared bases to reduce memory and computation costs. In contrast, MBF [2] exploits the input–output structures of neural networks to identify minimal "mini-blocks" that are easier to compute. More recently, AdaFisher [4] has proposed to retain only the diagonal entries of the Kronecker factors to guide gradient descent.

Our BD-KFAC approach can be viewed as an intermediate strategy between AdaFisher and MBF, aiming to balance curvature accuracy and resource efficiency.

3. Background

K-FAC is a second-order optimization method that approximates the inverse Fisher Information Matrix (FIM) using Kronecker-factored matrices. For a network with L layers, the FIM is first approximated as block-diagonal:

$$F \approx \mathrm{blockdiag}(F_1, F_2, \ldots, F_L),$$

where each block F_l corresponds to the parameters of layer l.

Each block is further factorized as a Kronecker product:

$$F_l \approx A_{l-1} \otimes G_l,$$

with $A_{l-1} = \mathbb{E}[a_{l-1}a_{l-1}^\top]$ the covariance of activations from layer $l-1$, and $G_l = \mathbb{E}[g_l g_l^\top]$ the covariance of gradients with respect to the pre-activations of layer l.

By properties of the Kronecker product, the inverse of each block satisfies

$$F_l^{-1} \approx A_{l-1}^{-1} \otimes G_l^{-1}.$$

Thus, a damped K-FAC parameter update can be written as

$$\nabla_{W_l} \;=\; -\,\alpha\,(G_l + \gamma I)^{-1}\,\nabla_{W_l}\mathcal{L}(w_t)\,(A_{l-1} + \gamma I)^{-1}, \tag{1}$$

where adding γI (Tikhonov regularization) stabilizes the inverses.

In practice, computing matrix inverses is expensive. A common strategy is to replace the inversion with eigendecomposition. Specifically, let

$$A_{l-1} = Q_A \Lambda_A Q_A^\top, \qquad G_l = Q_G \Lambda_G Q_G^\top,$$

where Q_A, Q_G are eigenvector matrices and Λ_A, Λ_G are diagonal matrices of eigenvalues. Then the preconditioned gradient can be computed as

$$V_1 = Q_G^\top \nabla_{W_l}\mathcal{L}(w_t)\, Q_A, \qquad V_2 = \frac{V_1}{\Lambda_G \Lambda_A^\top + \gamma}, \qquad (F_l + \gamma I)^{-1}\nabla_{W_l}\mathcal{L}(w_t) = Q_G\, V_2\, Q_A^\top,$$

where the division is elementwise.

4. Method

For gradient preconditioning in a given neural network layer, our goal is to obtain n block-diagonal approximations of A^{-1} and G^{-1} and substitute them into Eq. 1. By properties of block-diagonal matrices,

$$A^{-1} \;\approx\; \big(\mathrm{blockdiag}(A_1, \dots, A_n)\big)^{-1} = \mathrm{blockdiag}(A_1^{-1}, \dots, A_n^{-1}),$$

and analogously for G^{-1}.

Suppose the input feature dimension is d_{in}, which we partition into n groups $S_1, \dots, S_n$, and let a_{S_i} denote the subvector of activations indexed by S_i. After partitioning, we set

$$A \;\approx\; \mathrm{blockdiag}(A_1, \dots, A_n), \qquad A_i \;=\; \mathbb{E}\big[a_{S_i} a_{S_i}^\top\big].$$

Similarly, if we partition the pre-activation gradient coordinates into n groups $T_1, \dots, T_n$, then

$$G \;\approx\; \mathrm{blockdiag}(G_1, \dots, G_n), \qquad G_i \;=\; \mathbb{E}\big[g_{T_i} g_{T_i}^\top\big],$$

Therefore, as long as we reasonably partition the original a and g and reuse them within the existing Kronecker-factor computation, the original computational framework can be applied with almost no modification to the code.

4.1. Linear Layers

Let $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Partition input features into n (nearly) equal groups $S_1, \dots, S_n$ with the last group absorbing any remainder, and similarly partition output features into n groups $T_1, \dots, T_n$:

$$S_1 \dot\cup \cdots \dot\cup S_n = \{1, \dots, d_{\text{in}}\}, \qquad T_1 \dot\cup \cdots \dot\cup T_n = \{1, \dots, d_{\text{out}}\}.$$

Therefore, for activations of shape (batch_size, d_{in}), we take the slice (batch_size, S_1) corresponding to the indices S_1 as the sub-block activation and compute $A_i \;=\; \mathbb{E}\big[a_{S_i}\, a_{S_i}^\top\big]$. Similarly, for gradients with respect to the pre-activations of shape (batch_size, d_{out}), we take the slice (batch_size, T_i) corresponding to the indices T_i as the sub-block gradient and compute $G_i \;=\; \mathbb{E}\big[g_{T_i}\, g_{T_i}^\top\big]$.

4.2. Convolutional Layers

For a convolution with $W \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_h \times k_w}$, partition input channels into $S_1, \dots, S_n$ and output channels into $T_1, \dots, T_n$ (the last set absorbs any remainder). Using the standard patch (im2col) view, the activation-side blocks have effective feature width $|S_i|\, k_h k_w$.

For each activation block S_i, we first extract local receptive-field vectors $a_{S_i}(u,v) \in \mathbb{R}^{|S_i|k_h k_w}$ at every spatial location (u,v), optionally augment with a bias term, and then average over spatial positions. This yields

$$\bar{a}_{S_i} \;=\; \tfrac{1}{H'W'} \sum_{u=1}^{H'} \sum_{v=1}^{W'} a_{S_i}(u,v), \qquad A_i \;=\; \mathbb{E}\big[\,\bar{a}_{S_i}\,\bar{a}_{S_i}^\top\big].$$

For each gradient block T_i, let $g_{T_i}(u,v) \in \mathbb{R}^{|T_i|}$ denote the pre-activation gradients at spatial location (u,v) (restricted to channels in T_i). We average over spatial positions and form the second moment:

$$\bar{g}_{T_i} \;=\; \frac{1}{H'W'} \sum_{u=1}^{H'} \sum_{v=1}^{W'} g_{T_i}(u,v), \qquad G_i \;=\; \mathbb{E}\big[\,\bar{g}_{T_i}\,\bar{g}_{T_i}^\top\big].$$

4.3. Normalization Layers

For normalization layers such as LayerNorm and BatchNorm2d, we follow the formulation in AdaFisher [4] but implement our full-factor and block-diagonal variants.

On the activation side, we use a fixed 2×2 factor over the feature $[\hat{a},\,1]$ without input partitioning:

$$A \;=\; \mathbb{E}\left[\begin{bmatrix}\hat{a}\\1\end{bmatrix}\begin{bmatrix}\hat{a}\\1\end{bmatrix}^\top\right] \in \mathbb{R}^{2\times 2},$$

where $\hat{a}$ denotes the normalized activation.

On the gradient side, let g be the pre-activation gradients. For LayerNorm, we aggregate over all non-parameter axes (i.e., outside the normalized shape), while for BatchNorm2d we aggregate over spatial dimensions (H, W). Let $T_1, \ldots, T_n$ be a partition of the parameter (channel) dimension, and define

$$s_{T_i} \;=\; (g_{\text{param}})_{:,\,T_i} \in \mathbb{R}^{N\times|T_i|}, \qquad G \;\approx\; \operatorname{blockdiag}\Big(\mathbb{E}[s_{T_1}s_{T_1}^\top], \ldots, \mathbb{E}[s_{T_n}s_{T_n}^\top]\Big).$$

As in standard K-FAC, expectations are maintained with an exponential moving average for stability, and eigendecompositions are performed periodically to refresh the preconditioner.

4.4. Precomputing Block-Wise Inverses of A and G

As discussed around Eq. (1), using eigendecomposition is generally preferable to direct inversion on modern accelerators. However, applying the spectral form at *every* iteration requires computing

$$V_1 \;=\; Q_G^\top\,\nabla_{W_l}\mathcal{L}(w_t)\,Q_A, \qquad V_2 \;=\; \frac{V_1}{\Lambda_G\Lambda_A^\top + \gamma},$$

which introduces nontrivial per-step costs (the division is elementwise). Moreover, in distributed settings, repeatedly transmitting eigenvectors/eigenvalues (e.g., Q_A, Λ_A) triggers extra collective operations unless mitigated by fusion. To amortize both computation and communication, we *precompute* block-wise inverses of the K-FAC factors and reuse them across iterations.

For each block we form a damped eigendecomposition

$$A_i = Q_{A,i}\Lambda_{A,i}Q_{A,i}^\top, \qquad G_i = Q_{G,i}\Lambda_{G,i}Q_{G,i}^\top,$$

Table 1: Accuracy (%) on CIFAR-10/100. Best per column in **bold**.

	CIFAR-10	**CIFAR-100**						
Optimizer	ResNet-18	ResNet-34	CCT	DenseNet	FocalNet	ResNet-50	Swin-S	Swin-B
BD-KFAC	**94.53**	**77.32**	56.49	**78.11**	57.86	**77.88**	**61.47**	**63.10**
AdamW	94.40	76.33	44.33	76.28	56.56	77.26	59.44	62.61
Shampoo	94.02	76.68	36.34	77.59	**59.93**	77.58	61.33	62.33
AdaFisherW	94.00	75.92	56.59	77.23	47.65	76.85	56.24	57.45
K-FAC	93.70	76.49	55.64	77.04	53.77	76.95	60.84	62.65
Adafactor	85.25	53.66	**57.47**	51.95	40.95	52.07	43.87	47.10

where $Q_{A,i}$ and $Q_{G,i}$ denote the eigenvector matrices, and $\Lambda_{A,i}$ and $\Lambda_{G,i}$ are diagonal matrices of the corresponding eigenvalues. We then define the *precomputed* inverses

$$A_i^{-1}(\gamma) \;:=\; Q_{A,i}\,(\Lambda_{A,i}+\gamma I)^{-1}\,Q_{A,i}^\top, \qquad G_i^{-1}(\gamma) \;:=\; Q_{G,i}\,(\Lambda_{G,i}+\gamma I)^{-1}\,Q_{G,i}^\top.$$

Stacking across blocks yields a block-diagonal preconditioner,

$$A^{-1}(\gamma) \approx \operatorname{blockdiag}\left(A_1^{-1}(\gamma),\ldots,A_n^{-1}(\gamma)\right), \quad G^{-1}(\gamma) \approx \operatorname{blockdiag}\left(G_1^{-1}(\gamma),\ldots,G_n^{-1}(\gamma)\right),$$

which we then use for gradient preconditioning via Eq. 1. This precomputation inevitably introduces some approximation error (Appendix A), but our ablations show that the efficiency gains outweigh these errors in practice.

5. Experiments

5.1. Image Classification

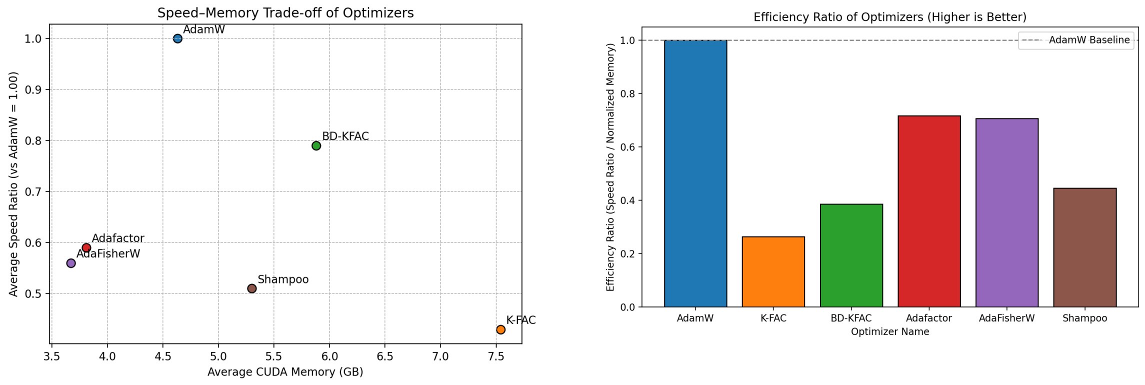
We evaluate the proposed method on image classification tasks, primarily focusing on CIFAR-100. Baseline methods include AdamW [10], Shampoo [1], AdaFisherW [4], and Adafactor [14]. To assess performance fairly, we adopt the wall-clock time metric as the primary evaluation criterion, which accounts for data loading, computation and communication costs except evaluation overhead. Specifically, we report results on ResNet-18 trained on CIFAR-10, as well as ResNet-34 [7], CCT-2/3x2 [6], DenseNet121 [8], FocalNet [15], ResNet-50 [7], Swin-S [9], and Swin-B [9] trained on CIFAR-100. The results are summarized in Table 1.

We observe that although our method performs less favorably on some lightweight ViT-style models, it consistently outperforms state-of-the-art baselines on the majority of medium- and large-scale CNNs as well as ViTs. In particular, even on architectures where it appears relatively disadvantaged, our approach still surpasses conventional K-FAC.

We further summarize the average runtime efficiency and the average peak GPU memory consumption of all methods in this study, using AdamW as the baseline, and present the results in Figure 1. As Figure 1 shows, our method, BD-KFAC, strikes a balance between full-matrix approximations and purely diagonal schemes.

6. Conclusion

In this work, we proposed a block-diagonal approximation of K-FAC as a second-order optimization method, aiming to strike a balance between convergence efficiency and resource efficiency by in-



(a) Speed–memory trade-off across optimizers

(b) Efficiency ratio normalized to AdamW memory

Figure 1: Comparison of optimizer efficiency: (a) scatter plot showing the trade-off between speed ratio and memory consumption; (b) bar chart showing efficiency ratio after normalizing AdamW's memory to 1.

terpolating between full-matrix second-order information and purely diagonal approximations. Experimental results on CIFAR-100 image classification demonstrate that, under the wall-clock time metric, our method outperforms state-of-the-art baselines on most model architectures, highlighting the effectiveness of block-diagonal approximations of K-FAC factors.

Future Work Although our PyTorch implementation currently executes block-wise eigendecompositions sequentially—thereby limiting further speedups—the blocks for a given layer (on both the activation and gradient sides) are inherently independent and can ingest their respective inputs in parallel. This observation makes our approach naturally compatible with the tensor-parallel design of Megatron-LM. As future work, we plan to integrate Block-Diagonal K-FAC with Megatron-LM's tensor parallelism to enable fully parallel block processing and thereby improve the training efficiency of large language models.

Acknowledgement

The author used GPT-5 to support academic English writing. After using this service, the author reviewed all the content and assumes full responsibility for the content of the publication.

This work is based on results obtained from the project, "Research and Development Project of the Enhanced infrastructures for Post-5G Information and Communication Systems" (JPNP20017), commissioned by the New Energy and Industrial Technology Development Organization (NEDO), and was supported by JSPS KAKENHI Grant Number JP22H00509, JST CREST JPMJCR24R4, the Multidisciplinary Cooperative Research Program in CCS, University of Tsukuba, and Fujitsu Limited.

This work was also supported by JST SPRING, Grant Number JPMJSP2124.

References

[1] Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020.

[2] Achraf Bahamou, Donald Goldfarb, and Yi Ren. A mini-block fisher method for deep neural networks. *arXiv preprint arXiv:2202.04124*, 2022.

[3] Sai Surya Duvvuri, Fnu Devvrit, Rohan Anil, Cho-Jui Hsieh, and Inderjit Dhillon. Caspr: Combining axes preconditioners through kronecker approximation for deep learning. In *Forty-first International Conference on Machine Learning*, 2024.

[4] Damien Martins Gomes, Yanlei Zhang, Eugene Belilovsky, Guy Wolf, and Mahdi S Hosseini. Adafisher: Adaptive second order optimization via fisher information. *arXiv preprint arXiv:2405.16397*, 2024.

[5] Roger B. Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 573–582. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/grosse16.html.

[6] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[8] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

[9] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *CoRR*, abs/2103.14030, 2021. URL https://arxiv.org/abs/2103.14030.

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[11] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

[12] Kazuki Osawa, Satoki Ishikawa, Rio Yokota, Shigang Li, and Torsten Hoefler. Asdl: A unified interface for gradient preconditioning in pytorch, 2023. URL https://arxiv.org/abs/2305.04684.

[13] J. Gregory Pauloski, Qi Huang, Lei Huang, Shivaram Venkataraman, Kyle Chard, Ian T. Foster, and Zhao Zhang. KAISA: an adaptive second-order optimizer framework for deep neural networks. *CoRR*, abs/2107.01739, 2021. URL https://arxiv.org/abs/2107.01739.

[14] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR, 2018. URL http://proceedings.mlr.press/v80/shazeer18a.html.

[15] Jianwei Yang, Chunyuan Li, Xiyang Dai, and Jianfeng Gao. Focal modulation networks. *Advances in Neural Information Processing Systems*, 35:4203–4217, 2022.

[16] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10665–10673, 2021.

[17] Jui-Nan Yen, Sai Surya Duvvuri, Inderjit Dhillon, and Cho-Jui Hsieh. Block low-rank preconditioner with shared basis for stochastic optimization. *Advances in Neural Information Processing Systems*, 36:17408–17419, 2023.

[18] Jihun Yun, Aurelie Lozano, and Eunho Yang. Adablock: Sgd with practical block diagonal matrix adaptation for deep learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2574–2606. PMLR, 2022.

Appendix A. Approximation Error from Precomputed Inverses

Recall that the exact K-FAC update with eigen-decomposition involves

$$V_2 = \frac{V_1}{\Lambda_G \Lambda_A^\top + \gamma},$$

that is, each eigendirection (i, j) of (G, A) is scaled by

$$\alpha_{ij}^{\text{exact}} = \frac{1}{d_{g,i}\, d_{a,j} + \gamma},$$

where $d_{g,i}, d_{a,j}$ denote eigenvalues of G and A, respectively. In contrast, when we precompute and store the damped inverses

$$A^{-1}(\gamma) = Q_A(\Lambda_A + \gamma I)^{-1} Q_A^\top, \qquad G^{-1}(\gamma) = Q_G(\Lambda_G + \gamma I)^{-1} Q_G^\top,$$

and then form the product $G^{-1}(\gamma)\, \nabla L\, A^{-1}(\gamma)$ at training time, the effective scaling on eigendirection (i, j) becomes

$$\alpha_{ij}^{\text{fact}} = \frac{1}{(d_{g,i} + \gamma)(d_{a,j} + \gamma)}.$$

Thus the discrepancy between the exact and factored forms is

$$\Delta_{ij} \;=\; \alpha_{ij}^{\text{fact}} - \alpha_{ij}^{\text{exact}} = \frac{1}{(d_{g,i}+\gamma)(d_{a,j}+\gamma)} - \frac{1}{d_{g,i}d_{a,j}+\gamma} = \frac{\gamma\,(1-d_{g,i}-d_{a,j})-\gamma^2}{(d_{g,i}+\gamma)(d_{a,j}+\gamma)(d_{g,i}d_{a,j}+\gamma)}.$$

Consequently, for any $\gamma \geq 0$ we obtain the bound

$$|\Delta_{ij}| \;\leq\; \frac{\gamma\big(|1-d_{g,i}-d_{a,j}|+\gamma\big)}{(d_{g,i}+\gamma)(d_{a,j}+\gamma)(d_{g,i}d_{a,j}+\gamma)}.$$

- As $\gamma \to 0$, the error behaves like $O\left(\frac{\gamma}{d_{g,i}^2 d_{a,j}^2}\right)$.
- As $\gamma \to \infty$, the error decays at rate $O(\frac{1}{\gamma})$.

Thus, precomputing A^{-1} and G^{-1} introduces a controllable bias that the effect is quite small when using smaller damping.

Appendix B. Detailed Experimental Setup

We use a unified training setup across all models. The global batch size is fixed to 256. For CNN architectures we set weight decay to 5×10^{-4}, and for ViT-based architectures to 1×10^{-3}. We employ a OneCycleLR scheduler, with the peak learning rate reached at 20% of total training steps. To ensure fairness under wall-clock time, we pre-determine the number of epochs per method on a single node, using AdamW at 200 epochs as the reference. The resulting epoch budgets are: K-FAC (86), BD-KFAC (98), Adafactor (118), AdaFisherW (112), and Shampoo (102).

For the second-order preconditioners: block-diagonal K-FAC (ours), K-FAC, and Shampoo—we use AdamW as the base optimizer and keep its hyperparameters (e.g., learning rate and weight decay) identical to the AdamW baseline. Curvature statistics are refreshed every 12 iterations and the preconditioner is updated every 120 iterations.

We implemented our BD-KFAC method on top of KAISA [13], an open-source distributed K-FAC framework. KAISA assigns the inverse and eigendecomposition tasks of different layers to different nodes. After computing the Kronecker factors, it performs an all-reduce to obtain global averages, and after inversion/eigendecomposition, the results are broadcast to all nodes to update the global preconditioner. Our K-FAC baseline is also based on this implementation, while the Shampoo baseline is taken from ASDL [12].

Appendix C. Ablation Study

We conduct 50-epoch experiments on ResNet-50 for CIFAR-100 in an eight-node distributed setting to investigate how the number of blocks and our *inverse precomputation* affect model accuracy, runtime efficiency, and GPU memory usage. The results are reported in Table 3.

In distributed settings, our tailored K-FAC framework substantially narrows the wall-clock gap between full K-FAC and block-diagonal K-FAC. Nevertheless, under the same number of epochs, the block-diagonal variant remains competitive in accuracy, often matching or surpassing the full method. Notably, the unpartitioned (block-num = 1) K-FAC achieves a clear runtime advantage over standard full K-FAC while maintaining nearly identical accuracy, highlighting the efficiency and stability of our inverse precomputation strategy.

Table 2: Additional optimizer/preconditioner hyperparameters used on top of the unified setup.

Method	**Hyperparameter Settings**
AdamW	lr=0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$
K-FAC / BD-KFAC	damping $= 2.8 \times 10^{-3}$, KL-clip $= 2.0 \times 10^{-3}$, block num $= 4$
Shampoo	damping $= 1 \times 10^{-5}$
AdaFisherW	$\beta = 0.9$, $\lambda = 1 \times 10^{-3}$, $\gamma = 0.8$, $t_{\text{cov}} = 100$
Adafactor	lr = 0.003, $\beta_{2,\text{decay}} = -0.8$, $\epsilon_1 = 1 \times 10^{-30}$, $\epsilon_2 = 1 \times 10^{-3}$, $d = 1.0$

Table 3: Ablation study on block number and inverse precomputation (ResNet-50 on CIFAR-100, 50 epochs, 8 nodes).

Setting	Accuracy (%)	Runtime (s)
K-FAC	76.59	132.150
block num = 1 (no split)	76.60	**120.476**
block num = 2	76.56	126.726
block num = 4	76.53	131.694
block num = 6	76.67	138.184
block num = 8	76.67	149.975
block num = 10	**76.80**	151.344