DOUBLE PRECONDITIONING: OPTIMIZATION FOR TEST-TIME PERFORMANCE, NOT VALIDATION LOSS

Anonymous authors
Paper under double-blind review

ABSTRACT

Many modern applications of deep learning involve training a neural network via a one-step prediction loss (e.g., L^2 regression, cross-entropy), but deploy the network by rolling out along its own predictions. Key examples include autoregressive language modeling, flow-based generative modeling, and robot policy learning. It is well-documented that these settings induce a phenomenon we call **test-time feedback** (TTF): the mismatch between the training/validation loss and downstream metrics of interest, such as task success rate and generation quality, which grows with task length. While data curation, architecture, and objective design have been proposed to combat train-test shift in TTF settings, this paper proposes *optimization* as a new design axis to mitigate error accumulation. Specifically, we introduce a new optimization paradigm called **double-preconditioning** (`DoPr`) uniquely tailored to the challenges of TTF. `DoPr` combines gradient-wise preconditioning, as in `Adam` and `Muon`, with activation-wise preconditioning (`AP`), such as in `KFAC`. We show that the addition of `AP` yields a *drop-in intervention* for increasing downstream model performance across a range of TTF settings. Interestingly, these gains in test-time performance *do not consistently accompany improvements in validation loss*, opening new questions about how to properly evaluate models trained with one-step supervised objectives.

Anonymized Code: [link]

1 INTRODUCTION

Recent years have seen a resurgence in the design of novel optimizers for deep learning, particularly via the use of "matrix-shaped" normalization or **preconditioning**. These optimizers adjust gradient updates by a "preconditioning" matrix to reweight the descent direction. The successes highlight improved *convergence time* on training or validation loss, such as `Shampoo` (Gupta et al., 2018; Anil et al., 2020; Shi et al., 2023) winning the AlgoPerf optimizer competition (MLCommons, 2024) as measured by an overall "holdout error per unit compute" metric, and `Muon` (Jordan et al., 2024) coming to prominence in greatly accelerating low "time to target validation loss" on NanoGPT pretraining and seeing adoption in training production-scale language models (Team et al., 2025).

These successes implicitly assume that low generalization error on the training objective implies good downstream performance, e.g. pass@k in the context of language reasoning. However, it has been widely noted that validation loss may not track natural performance metrics. The two diverge under a regime we call **test-time feedback** (TTF), ubiquitous in contemporary deep learning, in which one *trains on per-step supervised objectives*, while deployment involves *rolling out multiple steps according to the model's own predictions*. Key examples include: (1) **robot policy learning** where per-step (or per-chunk) imitation of expert actions along expert trajectories, and deployment involves rolling out using the learned policy's actions and (2) autoregressive **LLM training** in which models are trained via next-token prediction, yet deployed via autoregressive sampling. In these settings, sequential deployment can cause the distribution of inputs (e.g., context tokens in language modeling) to diverge from that of the training distribution, leading to distribution shift driven solely by the model's own prediction errors. This "TTF shift" is widely attested in both robotics (Ross et al., 2011; Simchowitz et al., 2025) and in long-horizon language generation (e.g., (Bengio et al., 2015; Ranzato et al., 2015; Song et al., 2023)). Motivated by these findings and recent achievements in deep-learning optimizer design, we ask:

> Is there a design space for lightweight, generally applicable interventions on deep learning optimizers that target improved **downstream performance?**

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

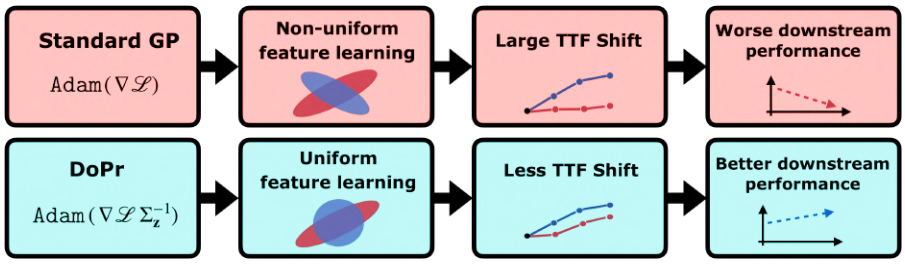


Figure 1: Standard optimizers can induce non-uniform feature learning (see Section 3.1), where small errors measured by validation loss $\mathcal{L}_{\text{val}}$ can cause outsized errors during deployment due to **test-time feedback** (TTF), inducing train-test mismatch leading to worsened downstream performance. We propose **Double Preconditioning** (`DoPr`), see Section 4, which targets more "uniform" feature learning for reducing TTF.

Contributions. **First**, we establish a rigorous connection between *distribution shift* induced by TTF and *imperfect feature learning* under today's popular optimizers that precondition updates solely using gradient statistics (e.g. Kingma & Ba (2015); Jordan (2025)). Specifically, we observe that when activations/inputs at each layer are non-isotropic, feature learning suffers, which induces errors that can disproportionately drive distribution shift under TTF. Moreover, these *cannot* be remedied by updates at later layers (Proposition 3.2).

Second, based on the above insights, we propose the **Double Preconditioning** (`DoPr`) framework, which couples an **activation-based preconditioner** (`AP`) using layer activations to compute a preconditioning matrix (e.g. Martens et al. (2018)) with more standard **gradient-based preconditioners** (`GP`), such as in `Adam` (Kingma & Ba, 2015; Loshchilov, 2017) or `Muon` (Jordan et al., 2024). `AP` encourages more uniformly accurate feature learning, mitigating the TTF distribution shift described above, whilst benefiting from the training speed and stability afforded by `GP`.

Whereas past variants of `AP` in the literature were introduced as standalone optimizers, `DoPr` shows how to incorporate `AP` as an **drop-in modification to any `GP` optimizer**, such as `Adam` and `Muon`, providing a stable training recipe and producing models with strong downstream performance. We develop an *invariance principle* under which `DoPr` updates can be systematically derived for any {architecture, `GP` } pairing, e.g., convolution and self-attention layers. Further, optimal hyperparameters can be reliably predicted by popular `GP` scaling heuristics (Yang & Littwin, 2023).

Finally, we carefully evaluate the **capabilities** of `DoPr` across a range of continuous-control, robotics, and language generation tasks. We find that `DoPr` consistently improves downstream performance—measured via natural, task-specific metrics across numerous applications— by intervening *purely on the optimizer*, without additional modifications to the data, training objective, or architecture.

Notation. We denote vector quantities by **bold** lower-case, and matrix/tensor quantities by **bold** upper-case. We use $\odot$ to denote element-wise (Hadamard) product, $\otimes$ for Kronecker product, and $\text{vec}(\cdot)$ the *column-major* vectorization operator. We reserve $\mathbf{W}$ to denote weights in their tensor shape, and $\boldsymbol{\theta} = \text{vec}(\mathbf{W})$ be the vectorized version. We use $\mathbb{E}[f(\mathbf{x})]$ to denote the expectation of $f(\mathbf{x})$. We reserve $\boldsymbol{\Sigma}$ for *uncentered* covariance matrices, e.g. $\Sigma_{\mathbf{x}} = \mathbb{E}[\mathbf{x}\mathbf{x}^\top]$. We denote layer-indexing with subscripts and (optimizer) iterates with superscripts, e.g., $\mathbf{W}_\ell^{(k)}$. Lastly, we use the index shorthand $[L] = \{1, \ldots, L\}$.

2 Learning under Test-Time Feedback

We study the problem of learning under **test-time feedback** (TTF) — where our goal is to learn a model that is deployed in feedback with its own generations (as in language modeling) or with the environment (as in robotics). We formalize TTF as a problem of behavior cloning (Pomerleau, 1988) in a Markov Decision Process (MDP). Consider a finite-horizon MDP with states $\mathbf{s}_t \in S$, actions $\mathbf{a}_t \in A$, transitions $\mathbf{s}_{t+1} \sim P(\mathbf{s}_t, \mathbf{a}_t)$, and total horizon T. *Models* or *policies* π are maps from $S \to \Delta(A)$, and we let $\mathbb{E}^\pi[\cdot]$ denote the expectation over actions $\mathbf{a}_t \sim \pi(\mathbf{s}_t)$ and state transitions $\mathbf{s}_{t+1} \sim P(\mathbf{s}_t, \mathbf{a}_t)$. In behavior cloning, we learn a policy $\pi_\theta : S \to \Delta(A)$ via demonstration sequences $(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)$. Given a per-example training loss $L_{\text{train}}(\pi; \mathbf{s}, \mathbf{a})$ measuring the distance between $\pi(\mathbf{s})$ and $\mathbf{a}$, and pairs of states and corresponding actions $(\mathbf{s}, \mathbf{a})$ (blue denoting

demonstrator/data distribution), we train a learned policy $\pi_{\boldsymbol{\theta}}$ by minimizing

$$\mathcal{L}_{\text{train}}(\pi_{\boldsymbol{\theta}}) = \sum_{(\mathbf{s},\mathbf{a})\in\text{data}} L_{\text{train}}(\pi_{\boldsymbol{\theta}}; \mathbf{s}, \mathbf{a}). \tag{2.1}$$

In the infinite data regime, and assuming $(\mathbf{s}_1, \mathbf{a}_1, \dots)$ are collected by a demonstrator π_{demo} deployed in the MDP, minimizing Eq. (2.1) approximately minimizes *validation loss*

$$\mathcal{L}_{\text{val}}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}^{\pi_{\text{demo}}}\left[\frac{1}{T}\sum_{t=1}^{T} L_{\text{train}}(\pi_\theta; \mathbf{s}_t, \mathbf{a}_t)\right]. \tag{2.2}$$

When there is no risk of confusion, we will abbreviate $L_{\text{train}} = L$ and $\mathcal{L}_{\text{val}} = \mathcal{L}$. Subsequently, π_θ is *deployed* in the MDP, and we evaluate its expected reward under the distribution of states $(\hat{\mathbf{s}}_1, \hat{\mathbf{a}}_1, \dots, \hat{\mathbf{s}}_T, \hat{\mathbf{a}}_T)$ induced by $\pi_{\boldsymbol{\theta}}$ under a trajectory-wise reward function $R(\cdot)$:

$$\mathcal{R}_{\text{test}}(\pi_{\boldsymbol{\theta}}) = \mathbb{E}^{\pi_{\boldsymbol{\theta}}}\left[R(\hat{\mathbf{s}}_1, \hat{\mathbf{a}}_1, \dots, \hat{\mathbf{s}}_T, \hat{\mathbf{a}}_T)\right]. \tag{2.3}$$

Language modeling. Language modeling with tokens x_t can be cast as an instance of TTF (Ouyang et al., 2022; Foster et al., 2024), where the states $\mathbf{s}_t = x_{1:t}$ denote the current context, actions $\mathbf{a}_t = x_{t+1}$ are the next token, and the dynamics are induced by concatenation: $\mathbf{s}_{t+1} = x_{1:t+1}$. The language model then produces tokens $x_{t+1} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid x_{1:t})$. Next-token prediction uses the cross entropy loss $L_{\text{train}}(\pi_{\boldsymbol{\theta}}; \mathbf{s}_t = x_{1:t}, \mathbf{a}_t = x_{t+1}) = -\log \pi_{\boldsymbol{\theta}}(x_{t+1} \mid x_{1:t})$. Typical choices of reward include success rate or pass@k (Chen et al., 2021b). **Robotic behavior cloning.** In robotics, $\mathbf{s}$ corresponds to the robot and environment state, and $\mathbf{a}$ to the robot actions. In practice, $\mathbf{s}$ is replaced by robot observations (e.g., pixels, tactile), and actions $\mathbf{a}$ may be short sequences or "action-chunks" (Chen et al., 2021a; Chi et al., 2023). Earlier works parameterize Gaussian policies $\pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{s}) = \mathrm{N}(\boldsymbol{\mu}_\theta(\mathbf{s}), \sigma^2\mathbf{I})$, motivating an L_2 training loss whereas modern works use generative model parameterizations (e.g. flows or diffusion (Chi et al., 2023; Pan et al., 2025)). Rewards can include task success or dense locomotive rewards.

Optimizer Preconditioning for Minimizing $\mathcal{L}_{\text{train}}$ in TTF. In our TTF formulation, $\mathcal{L}_{\text{train}}$ is a standard supervised-learning loss. We consider an optimization algorithm `alg` that produces iterates $\{\boldsymbol{\theta}^{(k)}\}_{k\geq 1}$ in order to minimize $\mathcal{L}_{\text{train}}$. Examples include mini-batch stochastic gradient descent (`SGD`), `Adam` (W), `Shampoo` (Gupta et al., 2018; Shi et al., 2023), and the recently popularized Muon optimizer (Jordan et al., 2024). Most popular deep learning optimizers can be viewed as *gradient-based preconditioners* (`GP`), where the update direction is shaped purely using gradient information. Many prominent members therein, such as `Adam`, `Shampoo`, `Muon` etc., are specifically structured approximations (e.g., diagonal, layer-wise) of the `AdaGrad` gradient-covariance $\sum_{k\geq 1} \nabla\mathcal{L}_{\text{train}}(\boldsymbol{\theta}^{(k)})\nabla\mathcal{L}_{\text{train}}(\boldsymbol{\theta}^{(k)})^\top$ (Duchi et al., 2011), and admit specific interpretations as gradient *whiteners* or *normalizers*. We provide full discussion in Appendix A. **Past work** has extensively studied how the choice of optimization improves performance only on the *validation loss* $\mathcal{L}_{\text{val}}$, both in terms of final iterate $\lim_{k\to\infty} \mathcal{L}_{\text{val}}(\boldsymbol{\theta}^{(k)})$, or the convergence rate, i.e., how quickly $L_{\text{val}}(\boldsymbol{\theta}^{(k)})$ approaches its limit. In this work, we consider a different problem statement:

Problem Statement 1

How do we design an optimizer to minimize $\mathcal{L}_{\text{train}}$, so as to maximize *downstream performance* of the resulting model $\pi_{\boldsymbol{\theta}}$ when evaluated at test-time, via $\mathcal{R}_{\text{test}}(\pi_{\boldsymbol{\theta}})$?

3 Distribution Shift under Test-Time Feedback

Whereas the validation loss $\mathcal{L}_{\text{val}}$ evaluates training loss under the distribution of states induced by π_{demo}, test-time reward $\mathcal{R}_{\text{test}}$ considers the distribution of states under $\pi_{\boldsymbol{\theta}}$. Due to sequential deployment in *feedback* with the MDP (e.g. through autoregressive generation in language modeling or with the environment in robotics), these two distributions do not agree (Figure 2). We call their consequent difference test-time feedback (TTF) shift. Let π_{demo} be the data-collection policy, and $\pi_{\boldsymbol{\theta}}$ a learned policy. Given a probability distance/divergence D (e.g. KL, Wasserstein Distance etc.), we define the **TTF shift** via $\mathrm{D}_t(\pi_{\boldsymbol{\theta}}) := \mathrm{D}(\mathbb{P}^{\pi_{\text{demo}}}_{\mathbf{s}_t}, \mathbb{P}^{\pi_{\boldsymbol{\theta}}}_{\hat{\mathbf{s}}_t})$, where $\mathbb{P}^{\pi}_{\mathbf{s}_t}$ denotes the marginal distribution of $\mathbf{s}_t \sim \mathbb{P}^{\pi}$. When TTF shift occurs, the distribution used to measure $\mathcal{L}_{\text{val}}$, $\mathbb{P}^{\pi_{\text{demo}}}_{\mathbf{s}_t}$, and that used for $\mathcal{R}_{\text{test}}$, $\mathbb{P}^{\pi_{\boldsymbol{\theta}}}_{\hat{\mathbf{s}}_t}$, differ, which has been shown in the literature to degrade performance $\mathcal{R}_{\text{test}}$. The effects of test-time distribution shifts has been extensively studied in the robotics and sequence-modeling literature (Ross et al., 2011; Bengio et al., 2015; Simchowitz et al., 2025); see Appendix A . Importantly, in contrast to general distribution shifts, TTF shift is **due to model error**

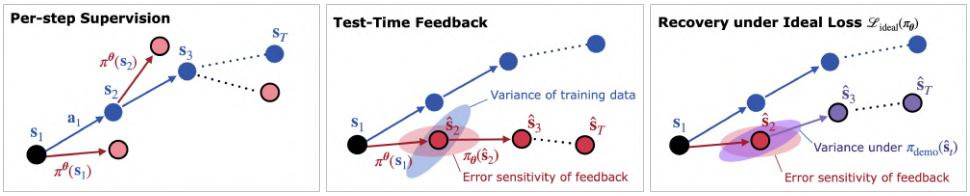


Figure 2: Many settings involve using per-step supervised objectives along training sequences. However, due to rolling out along the model's own predictions, mismatches between directions salient for $\mathcal{L}_{\text{val}}(\pi_{\boldsymbol{\theta}})$ versus $\mathcal{R}_{\text{test}}(\pi_{\boldsymbol{\theta}})$ cause TTF shift. Hypothetically, instead optimizing for directions salient for $\mathcal{L}_{\text{ideal}}(\pi_{\boldsymbol{\theta}})$—often unavailable for offline training—would induce smaller TTF shift.

propagated through sequential deployment. To reduce both TTF shift and its effects on downstream reward, past work has suggested that one strive to minimize an **idealized validation loss** with respect to actions from π_{demo}, but under the distribution of the *learned policy* $\pi_{\boldsymbol{\theta}}$ (Ross & Bagnell, 2010):

$$\mathcal{L}_{\text{ideal}}(\pi_{\boldsymbol{\theta}}) := \mathbb{E}^{\pi_{\boldsymbol{\theta}}}[\tfrac{1}{T}\textstyle\sum_{t=1}^{T} L_{\text{train}}(\pi_{\boldsymbol{\theta}}; \hat{\mathbf{s}}_t, \mathbf{a}_t^\star)], \quad \hat{\mathbf{s}}_t \sim \mathbb{P}^{\pi_{\boldsymbol{\theta}}}, \mathbf{a}_t^\star \sim \pi_{\text{demo}}(\cdot \mid \hat{\mathbf{s}}_t). \tag{3.1}$$

In particular, prior work (Ross & Bagnell, 2010; Foster et al., 2024) show that not only is $\mathcal{L}_{\text{ideal}}$ a good surrogate for $\mathcal{R}_{\text{test}}$, but that minimizing $\mathcal{L}_{\text{ideal}}$ would suffice to reduce TTF by implicitly correcting $\pi_{\boldsymbol{\theta}}$ towards π_{demo} when it veers off distribution (Figure 2(c)). However, minimizing $\mathcal{L}_{\text{ideal}}$ in practice is often challenging, due to its circular dependence on $\pi_{\boldsymbol{\theta}}$'s own distribution and the necessity of collecting π_{demo} actions on $\pi_{\boldsymbol{\theta}}$ states. [1] However, we are often *only* given data from $\mathbb{P}^{\pi_{\text{demo}}}$ (e.g., finetuning corpora or teleop demonstrations). Thus, we refine Problem Statement 1:

Problem Statement 2

Can we use **optimization preconditioning** to encourage minimization of $\mathcal{L}_{\text{ideal}}$ by mitigating TTF shift, even if we are only given data from π_{demo}?

3.1 TTF Shift and Imperfect Feature Learning

We consider **linear dynamical systems** (LDS) as a minimal example of TTF. We consider a LDS with state $\mathbf{s} \in \mathbb{R}^n$, actions $\mathbf{a} \in \mathbb{R}^m$, state transitions $\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{a}_t + \mathbf{w}_t$, where $\mathbf{w}_t \sim \mathrm{N}(0, \boldsymbol{\Sigma}_{\mathbf{w}})$ is process noise and $\mathbf{A}, \mathbf{B}$ are matrices of appropriate shape. We assume that the demonstrator $\pi_{\text{demo}}(\mathbf{s}) = \mathbf{K}_\star \mathbf{s}$ and imitator $\pi_{\boldsymbol{\theta}}(\mathbf{s}) = \mathbf{K}_\theta \mathbf{s}$ are both state feedback policies described by matrices in $\mathbb{R}^{m \times n}$, and the latter uses an L_2 training loss: $L_{\text{train}}(\pi_{\boldsymbol{\theta}}; \mathbf{s}, \mathbf{a}) = \|\mathbf{K}_\theta \mathbf{s} - \mathbf{a}\|_2^2 = \|(\mathbf{K}_\star - \mathbf{K}_\theta)\mathbf{s}\|_2^2 = L_{\text{train}}(\pi_{\boldsymbol{\theta}}; \mathbf{s})$, where the last equality notes that the $\mathbf{a}$ argument is redundant. Lastly, we consider a reward $R(\hat{\mathbf{s}}_1, \hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{s}}_T, \hat{\mathbf{a}}_T) = -\frac{1}{T}\sum_{t=1}^{T} L_{\text{train}}(\pi_{\boldsymbol{\theta}}; \hat{\mathbf{s}}_t) = -\frac{1}{T}\sum_{t=1}^{T} \|(\mathbf{K}_\star - \mathbf{K}_\theta)\hat{\mathbf{s}}_t\|^2$. In this setup, we can describe all relevant quantities in closed form:[2]

Proposition 3.1. *Define the state-covariance matrix* $\boldsymbol{\Gamma}_t(\mathbf{K}) = \sum_{s=0}^{t-1}(\mathbf{A} + \mathbf{B}\mathbf{K})^s \boldsymbol{\Sigma}_{\mathbf{w}} (\mathbf{A} + \mathbf{B}\mathbf{K})^{s\top}$, *and* $\bar{\boldsymbol{\Gamma}}_t(\mathbf{K}) = \frac{1}{t}\sum_{s=1}^{t} \boldsymbol{\Gamma}_s(\mathbf{K})$. *Then,* $\mathbb{P}_{\mathbf{s}_t}^{\pi_{\text{demo}}} = \mathrm{N}(0, \boldsymbol{\Gamma}_t(\mathbf{K}_\star))$, $\mathbb{P}_{\hat{\mathbf{s}}_t}^{\pi_{\boldsymbol{\theta}}} = \mathrm{N}(0, \boldsymbol{\Gamma}_t(\mathbf{K}_\theta))$, *and* $\mathcal{L}_{\text{val}}(\pi_{\boldsymbol{\theta}}) = \left\|(\mathbf{K}_\star - \mathbf{K}_\theta)\bar{\boldsymbol{\Gamma}}_T(\mathbf{K}_\star)^{1/2}\right\|_{\text{F}}^2$, $\mathcal{L}_{\text{ideal}}(\pi_{\boldsymbol{\theta}}) = -\mathcal{R}_{\text{test}}(\pi_{\boldsymbol{\theta}}) = \left\|(\mathbf{K}_\star - \mathbf{K}_\theta)\bar{\boldsymbol{\Gamma}}_T(\mathbf{K}_\theta)^{1/2}\right\|_{\text{F}}^2$. *If* D *denotes the Wasserstein-2 distance, we have* $\mathrm{D}_t(\pi_{\boldsymbol{\theta}}) = \|\Gamma_t(\mathbf{K}_\star)^{1/2} - \Gamma_t(\mathbf{K}_\theta)^{1/2}\|_{\text{F}}^2$.

Validation loss need not track downstream performance due to TTF. Proposition 3.1 characterizes TTF shift in terms of the state-covariance matrices induced by $\mathbf{K}_\star$, via $\boldsymbol{\Gamma}_t(\mathbf{K}_\star)$ (resp. $\mathbf{K}_\theta$ via $\boldsymbol{\Gamma}_t(\mathbf{K}_\theta)$). These same covariances weight the errors $\mathbf{K}_\star - \mathbf{K}_\theta$ in both $\mathcal{L}_{\text{val}}$ and $\mathcal{L}_{\text{ideal}}$. Thus, TTF and downstream performance depends not only on error magnitudes, but *in what directions* the error between policies $\mathbf{K}_\star - \mathbf{K}_\theta$ lies with respect to covariances. In Appendix B.5.1, we construct two linear policies $\pi_i(\mathbf{s}) = \mathbf{K}_{\pi_i}\mathbf{s}, i \in \{1, 2\}$ such that $L_{\text{val}}(\pi_1) \ll L_{\text{val}}(\pi_2)$, but $\mathcal{L}_{\text{ideal}}(\pi_1) \gg \mathcal{L}_{\text{ideal}}(\pi_2)$: $\mathbf{K}_{\pi_1} - \mathbf{K}_\star$ has error along small eigenspaces of $\boldsymbol{\Gamma}_T(\mathbf{K}_\star)$, but through feedback between $\mathbf{K}_{\pi_1}$ and the dynamics, these errors lead to large eigenvalues of $\boldsymbol{\Gamma}_T(\mathbf{K}_{\pi_1})$, driving up $\mathcal{L}_{\text{ideal}}$.

Understanding TTF from Feature Learning. To see how feature learning influences TTF, we *over*-parameterize the linear learner policy $\pi_{\boldsymbol{\theta}}$ as a product of linear layers $\pi_{\boldsymbol{\theta}}(\mathbf{s}) = \mathbf{K}_\theta \mathbf{s} = \mathbf{F}_\theta \mathbf{G}_\theta \mathbf{s}$,

[1] Adaptive data collection, such as the seminal DAgger (Ross et al., 2011), attempt to bridge this gap.

[2] See Appendix B.5 for proofs and further details of the results in this section.

where $\mathbf{F}_\theta \in \mathbb{R}^{m\times d}$, $\mathbf{G}_\theta \in \mathbb{R}^{d\times n}$; similarly write $\mathbf{K}_\star = \mathbf{F}_\star\mathbf{G}_\star$, and assume $\mathrm{rank}(\mathbf{K}_\star) = d$. A necessary condition for globally optimal imitation of $\pi_{\mathrm{demo}}(\cdot)$ is accurately recovering the (row-)span of $\mathbf{G}_\star$, e.g., measured by a subspace distance:

$$\mathrm{dist}(\mathbf{G}, \mathbf{G}_\star) := \left\|\mathcal{P}_{\mathbf{G}}\mathcal{P}_\star^\perp\right\|_{\mathrm{op}}, \tag{3.2}$$

where $\mathcal{P}_{\mathbf{G}}, \mathcal{P}_\star^\perp$ are orthogonal projections onto $\mathrm{rowsp}(\mathcal{P}_{\mathbf{G}}), \mathrm{rowsp}(\mathbf{G}_\star)^\perp$. Vanilla `GD`, by minimizing $\mathcal{L}_{\mathrm{val}}$, always ensures that $(\mathbf{G}_\theta - \mathbf{G}_\star)\mathbf{s}$ is small along directions $\mathbf{s}$ which align with large eigenvalues of $\mathbf{\Gamma}_T(\mathbf{K}_\star)$. However, large subspace distance $\mathrm{dist}(\mathbf{G}_\theta, \mathbf{G}_\star)$ means that there exist states $\mathbf{s}$ for which $\mathbf{G}_\star\mathbf{s}$ is non-zero, but $\mathbf{G}_\theta\mathbf{s}$ vanishes. Effectively, $\mathbf{G}_\theta$ "zeros out" certain directions of state-space, and therefore $\mathbf{F}_\theta\mathbf{G}_\theta$ will as well. Errors in these directions can compound, driving up TTF, and in the worst case, no adjustment of later layers can mitigate the errors made in $\mathbf{G}_\theta$:

Proposition 3.2 (Poor Feature Learning causes TTF Shift, Informal). *There exists* $(\mathbf{F}_1, \mathbf{G}_1)$, $(\mathbf{F}_2, \mathbf{G}_2)$, $(\mathbf{F}_\star, \mathbf{G}_\star)$ *and linear dynamics* $\mathbf{A}, \mathbf{B}, \mathbf{\Sigma}_{\mathbf{w}}$ *such that* $\mathcal{L}_{\mathrm{val}}(\mathbf{F}_1\mathbf{G}_1) \ll \mathcal{L}_{\mathrm{val}}(\mathbf{F}_2\mathbf{G}_2)$, *but* $\mathrm{dist}(\mathbf{G}_1, \mathbf{G}_\star) > \mathrm{dist}(\mathbf{G}_2, \mathbf{G}_\star)$, *and for any possible "last-layer"* $\mathbf{F}_\theta$, *we have* $\mathcal{L}_{\mathrm{ideal}}(\mathbf{F}_2\mathbf{G}_2) \ll \mathcal{L}_{\mathrm{ideal}}(\mathbf{F}_\theta\mathbf{G}_1)$.

Proposition 3.2 demonstrates that a model that appears better measured by in-distribution validation loss can hide significantly poorer feature learning, and can thus suffer vastly worse TTF, even assuming the downstream layers can be retrained. A natural question would be whether one can encourage better feature learning, as a separate consideration from validation loss performance.

4 Double Preconditioning (`DoPr`)

Toward addressing the pathologies of TTF, we propose the **double preconditioning** (`DoPr`) framework, which be summarized as applying layer-wise an *activation-covariance* preconditioner (`AP`) onto the gradient, then passing the `AP`-gradient into a *gradient preconditioner* (`GP`) of choice, such as in `Adam` or `Muon`. For concreteness, consider an L-layer feedforward network: $f_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{W}_L\phi(\mathbf{W}_{L-1}\cdots\phi(\mathbf{W}_1\mathbf{x})\cdots)$, where $\boldsymbol{\theta}$ is the concatenation of $\boldsymbol{\theta}_\ell = \mathrm{vec}(\mathbf{W}_\ell)$, $\ell \in [L]$. Denote the intermediate activations $\mathbf{z}_\ell \triangleq \phi(\mathbf{W}_{\ell-1}\mathbf{z}_{\ell-1}\cdots)$, $\mathbf{z}_1 \triangleq \mathbf{x}$ and pre-activations $\mathbf{h}_\ell = \mathbf{W}_\ell\mathbf{z}_\ell$.[3] The essential `DoPr` framework can be summarized as:

$$\mathbf{M} = \hat{\nabla}_{\mathbf{W}}\mathcal{L}(f_{\boldsymbol{\theta}})\,\hat{\mathbf{\Sigma}}_{\mathbf{z}}^{-1},\ \hat{\mathbf{\Sigma}}_{\mathbf{z}} \approx \mathbb{E}[\mathbf{z}\mathbf{z}^\top] \qquad (\texttt{ideal AP})$$

$$\mathbf{D} = \texttt{GP}(\mathbf{M}),\quad \mathbf{W}^{\mathrm{next}} \leftarrow \mathbf{W} - \eta\mathbf{D}, \qquad (\texttt{DoPr})$$

where $\hat{\nabla}_{\mathbf{W}}$ is the minibatch gradient, and $\hat{\mathbf{\Sigma}}_{\mathbf{z}}$ the batch empirical uncentered covariance of $\mathbf{z}$. We display the core algorithm in Algorithm 1, and discuss derivations for general architectures in Section 4.2. Full details and practical features are described in Appendix B. Whereas much of the prior literature has focused on the role of *gradient* pre-conditioned optimization for stabilizing training and improving validation loss, we will show that `DoPr`'s use of *activation* preconditioning mitigates TTF, and therefore **improves downstream model performance independently of validation loss performance**.

Algorithm 1: Double Preconditioning (Feedforward Layer, layer index ℓ suppressed)

Input: learning rate η, weight decay λ, damping γ
for $k = 1$ **to** K **do**
  Sample batch B, $|B| = n$.
  $\mathbf{G} \leftarrow \nabla_{\mathbf{W}}\widehat{\mathcal{L}}(\boldsymbol{\theta}^{(k-1)})$
  $\hat{\mathbf{\Sigma}}_{\mathbf{z}} \leftarrow \frac{1}{n}\sum_{i=1}^n \mathbf{z}_i^{(k-1)}\mathbf{z}_i^{(k-1)\top}$
  $\mathbf{M} \leftarrow \mathbf{G}\cdot\left(\hat{\mathbf{\Sigma}}_{\mathbf{z}} + \gamma\,\mathrm{tr}(\hat{\mathbf{\Sigma}}_{\mathbf{z}})\mathbf{I}\right)^{-1}$ (`AP`)
  $\mathbf{D} \leftarrow \texttt{GP}(\mathbf{M}, \texttt{**kwargs})$ (`DoPr`)
  $\mathbf{W}^{(k)} \leftarrow (1-\eta\lambda)\mathbf{W}^{(k-1)} - \eta\mathbf{D}$
end for

Gradient- v.s. Activation- preconditioning. We remark that activation preconditioning (`AP`) and many gradient preconditioners (`GP`'s) are motivated as different approximations to different curvature preconditioners; see Appendix A for an extensive account. Prior work has modeled `GP`'s (e.g., `Adam`, `Muon`) as performing (layer-wise) steepest descent with respect to a salient norm $\|\cdot\|$ (Bernstein & Newhouse, 2024; Pethick et al., 2025): $\mathbf{W}^{\mathrm{next}} = \mathbf{W} - \eta\arg\max_{\|\mathbf{G}\|\le 1}\langle\mathbf{G}, \nabla_{\mathbf{W}}\mathcal{L}\rangle$.[4] Just as vanilla `GD` can suffer unstable training in deep networks, the `AP` update requires stabilization to converge meaningfully on modern taskloads.

[3] We omit the layer index ℓ when it is clear from context.

[4] Practical features like momentum and weight decay can be incorporated with slight modifications (Pethick et al., 2025).

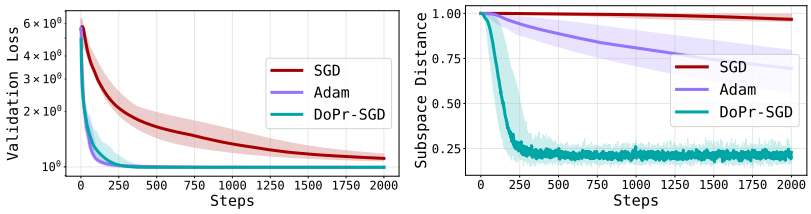


Figure 3: Mismatch between validation loss and feature learning. In-distribution validation loss converges and is even accelerated by GP. However, unless AP is applied, the feature subspace distance (3.2) converges poorly, which can exacerbate TTF (Proposition 3.2). Details in Appendix C.2.

This motivates the combination of AP with GP, yielding DoPr. We can express DoPr in the steepest descent GP framing by replacing the Euclidean inner product with the "activation-covariance metric":

$$\mathbf{W}^{\text{next}} = \mathbf{W} - \eta \arg\max_{\|\mathbf{G}\| \le 1} \langle \mathbf{G}, \nabla_{\mathbf{W}} \mathcal{L} \rangle_{\hat{\mathbf{\Sigma}}_{\mathbf{z}}^{-1}} = \mathbf{W} - \eta \arg\max_{\|\mathbf{G}\| \le 1} \langle \mathbf{G}, \nabla_{\mathbf{W}} \mathcal{L} \cdot \hat{\mathbf{\Sigma}}_{\mathbf{z}}^{-1} \rangle.$$

For example, setting $\|\cdot\| = \|\cdot\|_\infty$ as the entry-wise ℓ^∞-norm yields a Sign-Descent AP update ($\sim$Adam), and $\|\cdot\|_{\text{RMS}-\text{RMS}}$ recovers a Spectral-Descent AP update ($\sim$Muon).

4.1 Theoretical Motivation: AP Mitigates TTF via Isotropic Feature Learning

Proposition 3.2 therefore reveals that **non-uniform feature learning** at a given layer (i.e., larger $\text{dist}(\mathbf{G}_1, \mathbf{G}_\star)$) can **induce greater TTF shift and reduce downstream performance**, regardless of what is learned at subsequent layers (the choice of $\mathbf{F}_1$). We now argue that AP optimizers exhibit **more uniform feature learning** by correcting a bias in GD that occurs when inputs are non-isotropic, i.e. $\mathbb{E}^{\pi_{\text{demo}}}[\mathbf{s}_t \mathbf{s}_t^\top]$ is ill conditioned (Collins et al., 2021; Zhang et al., 2024). Taken together, these findings imply that **AP optimizers have the potential to mitigate TTF shift** and its consequences.

Proposition 4.1 (Informal, adapted from Zhang et al. (2024)). *Let $\eta > 0$ denote the step size, $\mathbf{G}_\theta^{\text{next,GD}}$ be $\mathbf{G}_\theta$ after a step of* full-batch *gradient step on $L_{\text{train}}(\mathbf{F}_\theta \mathbf{G}_\theta)$ with respect to $\mathbf{G}_\theta$, and let $\mathbf{G}_\theta^{\text{next,AP}}$ denote the same with the* AP *update. Then, setting $\mathbf{\Sigma}_{\mathbf{s}} = \bar{\mathbf{\Gamma}}_T(\mathbf{K}_\star)$, we have:*

$$\begin{aligned} \mathbf{G}_\theta^{\text{next,GD}} \mathcal{P}_\star^\perp &= (\mathbf{G}_\theta - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta \mathbf{G}_\theta \mathbf{\Sigma}_{\mathbf{s}}) \mathcal{P}_\star^\perp + \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\star \mathbf{G}_\star \mathbf{\Sigma}_{\mathbf{s}} \mathcal{P}_\star^\perp \\ \mathbf{G}_\theta^{\text{next,AP}} \mathcal{P}_\star^\perp &= (\mathbf{I} - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta) \mathbf{G}_\theta \mathcal{P}_\star^\perp. \end{aligned} \tag{4.1}$$

Consequently, omitting some technical details, when $\text{cond}(\bar{\mathbf{\Gamma}}_T(\mathbf{K}_\star)) \gg 1$,

$$\text{dist}(\mathbf{G}_\theta^{\text{next,GD}}, \mathbf{G}_\star) \approx \text{dist}(\mathbf{G}_\theta, \mathbf{G}_\star), \quad \text{dist}(\mathbf{G}_\theta^{\text{next,AP}}, \mathbf{G}_\star) \le \big(1 - \eta\, \lambda_{\min}(\mathbf{F}_\theta{}^\top \mathbf{F}_\theta)\big)\, \text{dist}(\mathbf{G}_\theta, \mathbf{G}_\star).$$

The proof of Proposition 4.1 can be found in Appendix B.4. The first statement exposes the *bias* of GD under non-isotropic covariance, and the non-convergence of GD and contraction of AP in feature space. Notably, under AP the resulting update (4.1) behaves *as if the features were isotropic* $\mathbf{\Sigma}_{\mathbf{s}} = \mathbf{I}$, which is uniquely where GD and AP experience the same feature learning, noting $\mathbf{G}_\star \mathbf{\Sigma}_{\mathbf{s}} \mathcal{P}_\star^\perp = \mathbf{G}_\star \mathcal{P}_\star^\perp = \mathbf{0}$. We consider the full-batch case for simplicity to show GD even suffers here; extensions to the minibatch case are straightforward. Notably, we show in Figure 3 that **pure GP (e.g. Adam, Muon) do not yield the same improvements**, even if the $\mathcal{L}_{\text{val}}$ convergence is seemingly accelerated. The pathology illustrated in Proposition 4.1 can be inductively applied for multi-layer networks, where under TTF, poor feature learning at early layers will destroy signal for all downstream layers.

4.2 Lifting AP to General Architectures: An Invariance Principle

In Proposition 4.1, we see that optimizers suffer from impaired feature learning, thus exacerbating TTF, when the activation covariances are ill-conditioned, and observed that AP mitigates this by feature-learning *as if the covariances were isotropic*. We extend this observation to yield a general **invariance principle** for generalizing DoPr to general network layers, such as convolutions and self-attention. We first consider the following thought experiment: consider arbitrary layerwise full-rank affine transformations:

$$\overline{\mathbf{z}}_\ell \triangleq \mathbf{A}_\ell \mathbf{z}_\ell, \overline{\mathbf{W}}_\ell \triangleq \mathbf{W}_\ell \mathbf{A}_\ell^{-1}, \ \ell \in [L], \tag{4.2}$$

and denote the resulting network output $f_{\bar{\boldsymbol{\theta}}}(\cdot)$, by construction, the pre-activations remain the same: $\overline{\mathbf{W}}\overline{\mathbf{z}} = \mathbf{W}\mathbf{z}$, and thus the network outputs are identical $f_{\bar{\boldsymbol{\theta}}}(\overline{\mathbf{x}}) = f_{\boldsymbol{\theta}}(\mathbf{x})$. Assume the loss gradient

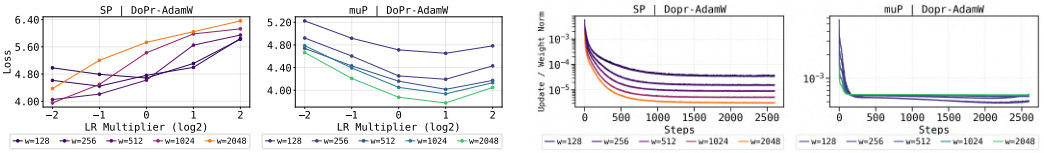


Figure 4: μP scaling behavior. Left: DoPr-AdamW's scaling trends under standard (SP) and AdamW's μP parameterizations on a GPT2 model. We find base AdamW's μP-scaling also enables hyperparameter transfer for DoPr-AdamW. **Right:** update-to-weight norm ratio scaling trend, under standard constant weight decay (SP) and AdamW's weight decay μP scaling. See Appendix C.4 for full details.

$\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is non-zero. Applying the backpropagation formula reveals that after a step of (full-batch) gradient descent: $\mathbf{W}^{\text{next}} = \mathbf{W} - \eta\,\mathbb{E}\big[\frac{\partial L}{\partial \mathbf{h}}\mathbf{z}^\top\big]$, whereas $\overline{\mathbf{W}}^{\text{next}} = \overline{\mathbf{W}} - \eta\,\mathbb{E}\Big[\frac{\partial L}{\partial \overline{\mathbf{h}}}\overline{\mathbf{z}}^\top\Big] = \mathbf{W}^{\text{next}}\mathbf{A}^{-1} + \eta\frac{\partial \mathcal{L}}{\partial \mathbf{W}}(\mathbf{A}^{-1} - \mathbf{A}^\top)$, such that the updated layer outputs are non-identical $\mathbf{W}_\ell^{\text{next}}\mathbf{z}_\ell \neq \overline{\mathbf{W}}_\ell^{\text{next}}\overline{\mathbf{z}}_\ell$ barring the special case of orthogonal $\mathbf{A}_\ell$ for all $\ell \in [L]$, and thus the updated models no longer have identical outputs: $f_{\boldsymbol{\theta}^{\text{next}}}(\mathbf{x}) \not\equiv f_{\bar{\boldsymbol{\theta}}^{\text{next}}}(\overline{\mathbf{x}})$. Consequently, the activation distributions propagated from the training distribution, e.g., $\Sigma_{\mathbf{s}} = \mathbb{E}^{\pi_{\text{demo}}}[\mathbf{s}_t\mathbf{s}_t^\top]$ versus $\Sigma_{\overline{\mathbf{s}}} = \mathbf{A}_1\Sigma_{\mathbf{s}}\mathbf{A}_1$, *impart a coordinate-dependent bias on the optimizer trajectory*, which Proposition 3.1 and Proposition 3.2 demonstrate can exacerbate TTF. Deriving AP for generic layer architectures boils down to the following principle: under an affine transformation on the input to the layer (and appropriate inverse transform to the applied weights) that preserve the layer output, **what is the update rule that renders the layer outputs of the updated network invariant to the transform**?

Proposition 4.2. *Consider a feedforward network, and consider the weights and activations* $\{(\mathbf{W}_\ell, \mathbf{z}_\ell)\}_{\ell\in[L]}$ *under non-degenerate layer-wise affine transforms defined in* (4.2)*. Then, given the following layer-wise update rule:* $\mathbf{W}^{\text{next}} = \mathbf{W} - \eta\nabla_{\mathbf{W}}\mathcal{L}(f_{\boldsymbol{\theta}})\boldsymbol{\Sigma}_{\mathbf{z}}^{-1}$, $\boldsymbol{\Sigma}_{\mathbf{z}} = \mathbb{E}[\mathbf{z}\mathbf{z}^\top]$ (AP)*, the updated weights satisfy* $\mathbf{W}_\ell^{\text{next}}\mathbf{z}_\ell = \overline{\mathbf{W}}_\ell^{\text{next}}\overline{\mathbf{z}}_\ell$ *for all* $\ell \in [L]$, $\mathbf{z}_1 = \mathbf{x}, \overline{\mathbf{z}}_1 = \mathbf{A}_1\mathbf{x}$.

In other words, for a given layer architecture the AP update is defined as one that induces invariance to affine transforms of the activation basis. Thus, AP updates for convolutional or self-attention layers can be easily derived; see Appendix C.3. We emphasize that the activation-invariance of AP is an *exact* property of the optimizer trajectory (see Figure 9) and Proposition 4.2 *does not* imply accelerated loss convergence.

4.3 Gradient-Preconditioning and Hyperparameter Scaling for DoPr

Determining favorable hyperparameters for large-scale training runs on a novel optimizer can prove challenging. This is a key motivation behind the "maximum-update parameterization" (μP) (Yang et al., 2022; Dey et al., 2025), where the broad goal is to make hyperparameter choices invariant across network scales (e.g., width and depth); see Appendix C.4 for full discussion. Once made scale-invariant, hyperparameters can be tuned at small-scale and zero-shot transferred to the large-scale run. Recent work has demonstrated that in practice μP is largely determined by coarse statistics such as the magnitude of the update direction (Yang et al., 2023; Liu et al., 2025a; Hong & Wang, 2025). Thus, in addition to the practical stabilizing and accelerating properties of GPs, a key benefit of using GPs in DoPr is as follows: whereas deriving scaling rules for a new optimizer can often be laborious, the *normalizing* property of GP trivializes this process for DoPr.

Observation 4.3. Normalization by definition returns outputs of the same norm regardless of input. Thus, substituting the raw gradient with the AP gradient does not affect the magnitude of the update direction. Consequently, DoPr scaling rules can be ported directly from established rules based on the GP of choice, e.g. Adam, Muon, Shampoo etc. (Everett et al., 2024; Qiu et al., 2025).

We exhibit the immediate transfer of hyperparameter scaling rules in Figure 4, where we show learning rate and weight decay scaling transfer just using the base GP's existing scaling rules.

5 Capabilities

In Section 4, we derive DoPr to address the poor feature learning of standard optimizers in TTF settings, with the promise of inducing better downstream behavior. To demonstrate this, we run experiments on diverse tasks and metrics across continuous control, robot policy learning, and LLM training. Compared to settings like pre-training and RLVR, we focus on **TTF settings** that learn from off-policy (offline) data, and performance is measured on terminal task performance; see

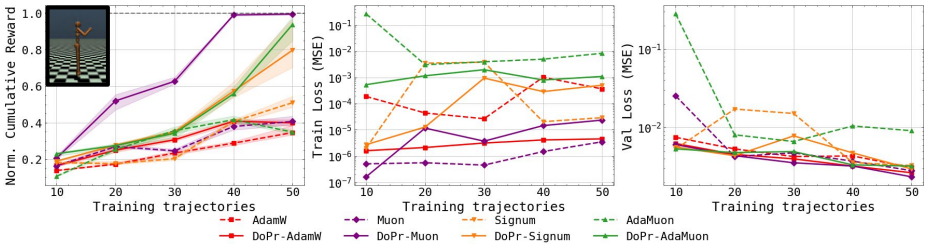


Figure 5: Humanoid-v5 `DoPr` performance across `AdamW`, `Muon`, `Signum`, and `AdaMuon`. `DoPr` variants attain higher terminal reward which does not consistently correlate with train or validation loss improvements.

Appendix C.1. As noted in Section 4.2, `DoPr` **need not improve the train/val loss convergence** compared to the base `GP`, which we will revisit across our experiments. We provide an account of the (amortizable) computational overhead in Appendix B.6, and provide in Appendix C.1 general operating guidelines of our optimizer used in the ensuing experiments. Across all experiments, we ensure **optimal hyperparameters are swept via standard validation loss criteria**, reflecting practice where hyperparameters must be determined before observing terminal performance.

5.1 Setting the Stage: Drop-in for Continuous Control

We first evaluate `DoPr` for imitation learning in Gymnasium (Towers et al., 2025), which precisely aligns with our formalism of behavior cloning in MDPs. Here, we focus on the `Humanoid-v5` task and refer to Appendix C.5 for full details and further experiments. We consider four `GP` primitives: `Adam`, `Muon`, `Signum`, and `AdaMuon`, that roughly cover instantaneous versus momentumized entrywise- and matrix-based normalization. We use a residual MLP architecture (He et al., 2015), and conduct a full sweep over the optimizer hyperparameters and report the results over independent evaluation trajectories and seeded training runs. We observe from Figure 5 that: 1. different base `GP`s all have similar terminal rewards, 2. `DoPr` always improves terminal reward compared to the baseline counterparts, 3. the train/val losses of `DoPr` variants are not always better than the base `GP`.

5.2 Image-Based Robot Policy Learning

We evaluate `DoPr` on pixel-based imitation learning with generative policies on `Robomimic` tasks. We focus on challenging tasks `Tool-Hang` Proficient-Human `(PH)` and `Transport (PH)`, which are not solvable to $\sim 100\%$ under modern BC recipes (Pan et al., 2025), and test complementary capabilities: dexterous precision in `Tool-Hang` and long-horizon coordination in `Transport`. Following Pan et al. (2025), we train a flow-based policy (Lipman et al., 2023) with a U-Net backbone (Chi et al., 2023). Tuned hyperparameters are listed in Tables 4 and 5, and full experiment details are in Appendix C.6. We report the best checkpoint success rate across training in Figure 6. We observe that `DoPr` invariably improves the task success rate compared to the base `GP`s `AdamW` and `Muon`, while yielding worse or equal training loss—see Figure 14. Notably, we see neither base `GP` is better than the other on both tasks simultaneously.

5.3 Language Models

We finetune an LLM for mathematical reasoning, where training optimizes token likelihood but task performance depends on sequence accuracy. Appendix C.7 contains full experimental details.

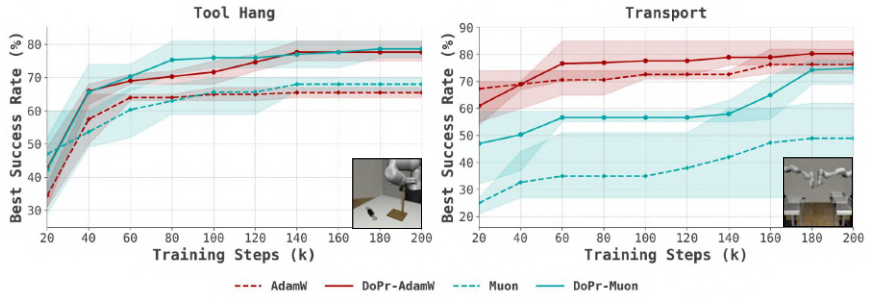


Figure 6: `Tool Hang (PH)` and `Transport (PH)` Best Success Rate for `AdamW`, `Muon`, and `DoPr` variants. Each curve shows the min/median/max over 3 random seeds. `DoPr`-variants outperform their baselines.

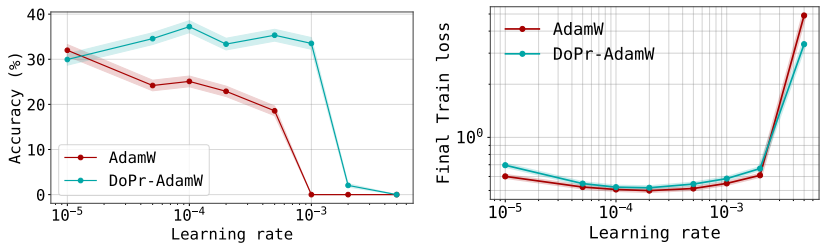


Figure 7: `GSM8K` **3B SFT sweep**. Peak `GSM8K` accuracy across training steps vs. learning rate, comparing `AdamW` with `DoPr-AdamW`. The comparison between `Muon` and `DoPr-Muon` is provided in Appendix C.7.1.

3B Supervised Fine-tuning. To characterize the behavior of `DoPr`, we first run a smaller-scale SFT experiment with the `Llama-3.2-3B` base model (AI@Meta, 2024; Grattafiori et al., 2024), using `LoRA` (Hu et al., 2022) on a 100K-sample subset of OpenMathInstruct-2 (Toshniwal et al., 2024) for one epoch. This experiment probes learning-rate sensitivity, sample efficiency, and the relationship between downstream accuracy and token-level training loss. We report our results in Figure 7. Across most learning rates, `DoPr` improves peak `GSM8K` performance. Notably, these improvements are *not explained by lower token-level training loss*: the final token-level training loss of `DoPr` is comparable to, or higher than, that of the base `GP`. We report similar trends for `Muon` (Figure 15) in Appendix C.7.1.

8B Supervised Fine-tuning: TTF at Larger Scale. Following the core SFT setup of `OpenMathInstruct-2` (Toshniwal et al., 2024), we fine-tune `Llama-3.1-8B` (AI@Meta, 2024; Grattafiori et al., 2024) on the `OpenMathInstruct-2` `train_1M` split for two epochs. We sweep learning rates for `AdamW` and `DoPr-AdamW`, and evaluate the final checkpoints on `GSM8K`, `GSM8K-CoT`, and `MATH-500`. We examine in Figure 8 how validation loss corresponds to downstream performance. For `DoPr-AdamW`, lower validation NLL generally corresponds to stronger downstream accuracy, especially on `GSM8K-CoT` and `MATH-500`. This is desirable as one would hope for a given recipe a model with the best validation loss also has the best performance. However, this is **not true** for baseline `AdamW`. For larger learning rates, `AdamW`-trained models predictably improve validation loss but task performance (e.g., `GSM8K`) degrades. This illustrates our thesis that, without intervention, validation loss can be a poor proxy for full-sequence performance. Finally, `DoPr-AdamW` also reduces cross-task conflict: optimal learning rates on `GSM8K-CoT` are also (near-)optimal on `MATH-500`, whereas no `AdamW` checkpoints are simultaneously near-optimal for `GSM8K` and `MATH-500` performance. This suggests that, as previewed in Section 4.1, `DoPr` encourages more robust feature learning. Additional results can be found in Appendix C.7.2.

Generative Modeling. We also view generative modeling as a TTF setting and report `DoPr` results on score-based ImageNet-256 generation in Appendix C.8 due to space constraints.

6 Discussion

Toward the ultimate goal of optimizing for downstream performance, we: 1. introduce Test-Time Feedback as a unified setting for many modern applications, 2. identify the mismatch between features that accelerate training convergence versus those sensitive under TTF, 3. prescribe Double Preconditioning (`DoPr`) that combines Activation Preconditioning (`AP`) to equalize feature learning and Gradient Preconditioning (`GP`) to stabilize training. Importantly, we find that improved downstream performance may not accompany improved train/val loss, suggesting that there remains a design space for improving deep learning optimization complementary to accelerating loss convergence.

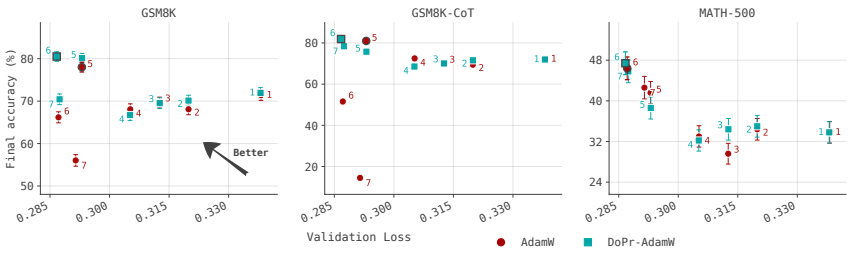


Figure 8: 8B SFT sweep. We plot final accuracy on `GSM8K`, `GSM8K-CoT`, and `MATH-500` against validation loss (NLL), computed on a 10K held-out subset of the training data. Each point is a checkpoint from a learning-rate sweep with labels indexing by learning rate size (small to large). Error bars indicate one standard error.

AI USE STATEMENT

AI was used for experiment infrastructure, sweep set-ups, and plot iterations. AI was used for verification of some supplementary mathematical results. Minimal AI was used in paper writing. No AI was used during the ideation process, algorithm design, theory-building, and literature search.

REPRODUCIBILITY STATEMENT

All mathematical results have corresponding proofs in Appendix B, and all external results are cited. Full algorithmic details and experiment set-ups are described throughout the main paper as well as Appendix C and D. Code for reproducing the experiments is attached after the abstract as a link to an anonymized repository.

REFERENCES

Kwangjun Ahn, Noah Amsel, and John Langford. Dion2: A simple method to shrink matrix in muon. *arXiv preprint arXiv:2512.16928*, 2025a.

Kwangjun Ahn, Byron Xu, Natalie Abreu, Ying Fan, Gagik Magakyan, Pratyusha Sharma, Zheng Zhan, and John Langford. Dion: Distributed orthonormalized updates, 2025b.

AI@Meta. Llama 3 model card, 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

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

Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2000.

Ehsan Amid, Rohan Anil, and Manfred Warmuth. Locoprop: Enhancing backprop via local loss optimization. In *International Conference on Artificial Intelligence and Statistics*, 2022.

Kang An, Yuxing Liu, Rui Pan, Yi Ren, Shiqian Ma, Donald Goldfarb, and Tong Zhang. Asgo: Adaptive structured gradient optimization. *Advances in Neural Information Processing Systems*, 38:126775–126814, 2026.

Maksym Andriushchenko, Dara Bahri, Hossein Mobahi, and Nicolas Flammarion. Sharpness-aware minimization leads to low-rank features. *Advances in Neural Information Processing Systems*, 36: 47032–47051, 2023.

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

Dara Bahri, Hossein Mobahi, and Yi Tay. Sharpness-aware minimization improves language model generalization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7360–7371, 2022.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Frederik Benzing. Gradient descent on neurons and its link to approximate second-order optimization. In *International Conference on Machine Learning*, 2022.

Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in llm pre-training, 2025. URL https://arxiv.org/abs/2505.13738.

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. *arXiv preprint arXiv:2409.20325*, 2024.

Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. In *Forty-second International Conference on Machine Learning*, 2025.

Adam Block, Dylan J Foster, Akshay Krishnamurthy, Max Simchowitz, and Cyril Zhang. Butterfly effects of SGD noise: Error amplification in behavior cloning and autoregression. In *The Twelfth International Conference on Learning Representations*, 2024.

David E Carlson, Edo Collins, Ya-Ping Hsieh, Lawrence Carin, and Volkan Cevher. Preconditioned spectral descent for deep learning. *Advances in neural information processing systems*, 28, 2015.

Roger Creus Castanyer, Johan Obando-Ceron, Lu Li, Pierre-Luc Bacon, Glen Berseth, Aaron Courville, and Pablo Samuel Castro. Stable gradients for stable learning at scale in deep reinforcement learning. *arXiv preprint arXiv:2506.15544*, 2025.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021a.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021b.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. Exploiting shared representations for personalized federated learning. In *International Conference on Machine Learning*, 2021.

Michael Crawshaw, Chirag Modi, Mingrui Liu, and Robert M Gower. An exploration of non-euclidean gradient descent: Muon and its many variants. *arXiv preprint arXiv:2510.09827*, 2025.

Alex Damian, Jason Lee, and Mahdi Soltanolkotabi. Neural networks can learn representations with gradient descent. In *Conference on Learning Theory*, 2022.

Felix Dangel, Bálint Mucsányi, Tobias Weber, and Runa Eschenhagen. Kronecker-factored approximate curvature (kfac) from scratch. *arXiv preprint arXiv:2507.05127*, 2025.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Damek Davis and Dmitriy Drusvyatskiy. When do spectral gradient updates help in deep learning? *arXiv preprint arXiv:2512.04299*, 2025.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: Completep enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618*, 2025.

Zhehang Du and Weijie Su. The newton-muon optimizer. *arXiv preprint arXiv:2604.01472*, 2026.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

Runa Eschenhagen, Alexander Immer, Richard Turner, Frank Schneider, and Philipp Hennig. Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, 36:33624–33655, 2023.

Katie E Everett, Lechao Xiao, Mitchell Wortsman, Alexander A Alemi, Roman Novak, Peter J Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12666–12700. PMLR, 21–27 Jul 2024.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

Dylan J Foster, Adam Block, and Dipendra Misra. Is behavior cloning all you need? understanding horizon in imitation learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Kevin Frans, Sergey Levine, and Pieter Abbeel. A stable whitening optimizer for efficient neural network training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

Thomas Frerix, Thomas Möllenhoff, Michael Moeller, and Daniel Cremers. Proximal backpropagation. In *International Conference on Learning Representations*, 2018.

Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, et al. Language models scale reliably with over-training and on downstream tasks. In *International Conference on Learning Representations*, volume 2025, pp. 67661–67682, 2025.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. *Advances in neural information processing systems*, 31, 2018.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Linearized two-layers neural networks in high dimension. *The Annals of Statistics*, 49(2):1029–1054, 2021a.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When do neural networks outperform kernel methods? *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12), 2021b.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582. PMLR, 2016.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Naicheng He, Kaicheng Guo, Arjun Prakash, Saket Tiwari, Ruo Yu Tao, Tyrone Serapio, Amy Greenwald, and George Konidaris. Spectral collapse drives loss of plasticity in deep continual learning. *arXiv preprint arXiv:2509.22335*, 2025.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 30016–30030, 2022.

Euijin Hong and Guannan Qu. Activation-keyed momentum: An anisotropic momentum update via the delta rule, 2026.

Letong Hong and Zhangyang Wang. On the provable separation of scales in maximal update parameterization. In *Forty-second International Conference on Machine Learning*, 2025.

Stefan Horoi, Guy Wolf, Eugene Belilovsky, and Gintare Karolina Dziugaite. Less is more: Under-training experts improves model upcycling. *arXiv preprint arXiv:2506.14126*, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Zarif Ikram, Arad Firouzkouhi, Stephen Tu, Mahdi Soltanolkotabi, and Paria Rashidinejad. Crispedit: Low-curvature projections for scalable non-destructive llm editing. *arXiv preprint arXiv:2602.15823*, 2026.

Satoki Ishikawa and Ryo Karakida. On the parameterization of second-order optimization effective towards the infinite width. *arXiv preprint arXiv:2312.12226*, 2023.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Keller Jordan. cifar-10 airbench, 2025. URL https://github.com/KellerJordan/cifar10-airbench.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8077–8083. IEEE, 2019.

Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from Adam to SGD. *arXiv preprint arXiv:1712.07628*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Atli Kosson, Bettina Messmer, and Martin Jaggi. Rotational equilibrium: How weight decay balances learning across neural networks, 2024. URL https://arxiv.org/abs/2305.17212.

Atli Kosson, Jeremy Welborn, Yang Liu, Martin Jaggi, and Xi Chen. Weight decay may matter more than μp for learning rate transfer in practice. In *OPT 2025: Optimization for Machine Learning*, 2025. URL https://openreview.net/forum?id=RVxJ9rWbZk.

Dmitry Kovalev. Understanding gradient orthogonalization for deep learning via non-euclidean trust-region optimization. *arXiv preprint arXiv:2503.12645*, 2025.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

Atharva Kulkarni, Jacob Mitchell Springer, Arjun Subramonian, and Swabha Swayamdipta. Disentangling geometry, performance, and training in language models. *arXiv preprint arXiv:2602.20433*, 2026.

Frederik Kunstner, Philipp Hennig, and Lukas Balles. Limitations of the empirical fisher approximation for natural gradient descent. *Advances in neural information processing systems*, 32, 2019.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pp. 143–156. PMLR, 2017.

Xi-Lin Li. Preconditioned stochastic gradient descent. *IEEE transactions on neural networks and learning systems*, 29(5):1454–1466, 2017.

Xi-Lin Li. Preconditioner on matrix lie group for sgd. *arXiv preprint arXiv:1809.10232*, 2018.

Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. Normuon: Making muon more efficient and scalable. *arXiv preprint arXiv:2510.05491*, 2025.

Wu Lin, Scott C Lowe, Felix Dangel, Runa Eschenhagen, Zikun Xu, and Roger B Grosse. Understanding and improving shampoo and soap via kullback-leibler minimization. *arXiv preprint arXiv:2509.03378*, 2025.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Andy Zeyi Liu, Elliot Paquette, and John Sous. Spectral lens: Activation and gradient spectra as diagnostics of llm optimization. *arXiv preprint arXiv:2605.05683*, 2026a.

Jiashun Liu, Zihao Wu, Johan Obando Ceron, Pablo Samuel Castro, Aaron Courville, and Ling Pan. Measure gradients, not activations! enhancing neuronal activity in deep reinforcement learning. *Advances in Neural Information Processing Systems*, 38:79546–79571, 2026b.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025a.

Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023.

Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Max Du, and Chelsea Finn. Bidirectional decoding: Improving action chunking via closed-loop resampling. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Yuxing Liu, Jianyu Wang, and Tong Zhang. Optimizer-model consistency: Full finetuning with the same optimizer as pretraining forgets less. *arXiv preprint arXiv:2605.06654*, 2026c.

I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Kevin Luk and Roger Grosse. A coordinate-free construction of scalable natural gradient. *arXiv preprint arXiv:1808.10340*, 2018.

Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in reinforcement learning. *arXiv preprint arXiv:2206.02126*, 2022.

Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *European Conference on Computer Vision*, pp. 23–40. Springer, 2024.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, 2015.

James Martens, Jimmy Ba, and Matt Johnson. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*, 2018.

Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and the double descent curve. *Communications on Pure and Applied Mathematics*, 75 (4):667–766, 2022.

AI Meta. Introducing meta llama 3: The most capable openly available llm to date. *Meta AI*, 2(5):6, 2024.

MLCommons. Announcing the results of the inaugural AlgoPerf: Training algorithms benchmark competition, 2024. URL https://mlcommons.org/2024/08/mlc-algoperf-benchmark-competition/.

Behrad Moniri, Donghwan Lee, Hamed Hassani, and Edgar Dobriban. A theory of non-linear feature learning with one gradient step in two-layer neural networks. In *International Conference on Machine Learning*, 2024.

Yann Ollivier. Riemannian metrics for neural networks i: feedforward networks. *arXiv preprint arXiv:1303.0818*, 2013.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The adEMAMix optimizer: Better, faster, older. In *The Thirteenth International Conference on Learning Representations*, 2025.

Chaoyi Pan, Giri Anantharaman, Nai-Chieh Huang, Claire Jin, Daniel Pfrommer, Chenyang Yuan, Frank Permenter, Guannan Qu, Nicholas Boffi, Guanya Shi, et al. Much ado about noising: Dispelling the myths of generative robotic control. *arXiv preprint arXiv:2512.01809*, 2025.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

Francisco Patitucci and Aryan Mokhtari. Adaptive optimization via momentum on variance-normalized gradients. *arXiv preprint arXiv:2602.10204*, 2026.

J Gregory Pauloski, Qi Huang, Lei Huang, Shivaram Venkataraman, Kyle Chard, Ian Foster, and Zhao Zhang. Kaisa: an adaptive second-order optimizer framework for deep neural networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2021.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained LMOs. In *Forty-second International Conference on Machine Learning*, 2025.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

Shikai Qiu, Zixi Chen, Hoang Phan, Qi Lei, and Andrew Gordon Wilson. Hyperparameter transfer enables consistent gains of matrix-preconditioned optimizers across scales. *arXiv preprint arXiv:2512.05620*, 2025.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, 2010.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Nicolas Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. *Advances in neural information processing systems*, 20, 2007.

Samuel Schapiro and Han Zhao. Towards understanding the role of sharpness-aware minimization algorithms for out-of-distribution generalization. *arXiv preprint arXiv:2412.05169*, 2024.

Robin M Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley-benchmarking deep learning optimizers. In *International Conference on Machine Learning*, 2021.

Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International conference on machine learning*, pp. 4596–4604. PMLR, 2018.

Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat. A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023.

Chongjie Si, Debing Zhang, and Wei Shen. Adamuon: Adaptive muon optimizer. *arXiv preprint arXiv:2507.11005*, 2025.

Max Simchowitz, Daniel Pfrommer, and Ali Jadbabaie. The pitfalls of imitation learning when actions are continuous. In Nika Haghtalab and Ankur Moitra (eds.), *Proceedings of Thirty Eighth Conference on Learning Theory*, volume 291 of *Proceedings of Machine Learning Research*, pp. 5248–5351. PMLR, 2025.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 32145–32168. PMLR, 2023.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *International Conference on Machine Learning*, pp. 32211–32252. PMLR, 2023.

Jacob Mitchell Springer, Sachin Goyal, Kaiyue Wen, Tanishq Kumar, Xiang Yue, Sadhika Malladi, Graham Neubig, and Aditi Raghunathan. Overtrained language models are harder to fine-tune. In *Forty-second International Conference on Machine Learning*, 2025.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Weijie Su. Isotropic curvature model for understanding deep learning optimization: Is gradient orthogonalization optimal? *arXiv preprint arXiv:2511.00674*, 2025.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural Networks for Machine Learning*, 4(2):26, 2012.

Alexander Tong, Kilian FATRAS, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2025. URL https://arxiv.org/abs/2407.17032.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M Kakade. Soap: Improving and stabilizing Shampoo using Adam. In *OPT 2024: Optimization for Machine Learning*, 2024.

Xi Wang and Laurence Aitchison. How to set adamw's weight decay as you scale model and dataset size. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=IszVnczhfz.

Ishaan Watts, Catherine Li, Sachin Goyal, Jacob Mitchell Springer, and Aditi Raghunathan. Sharpness-aware pretraining mitigates catastrophic forgetting. *arXiv preprint arXiv:2605.02105*, 2026.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 2017.

Lechao Xiao. Rethinking conventional wisdom in machine learning: From generalization to scaling, 2025. URL https://arxiv.org/abs/2409.15156.

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *International conference on machine learning*, pp. 5393–5402. PMLR, 2018.

Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. *Advances in neural information processing systems*, 32, 2019.

Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. *arXiv preprint arXiv:2006.14548*, 2020.

Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, 2021.

Greg Yang and Etai Littwin. Tensor programs ivb: Adaptive optimization in the infinite-width limit. *arXiv preprint arXiv:2308.01814*, 2023.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs VI: Feature learning in infinite depth neural networks. In *The Twelfth International Conference on Learning Representations*, 2024.

Thomas T. Zhang, Daniel Pfrommer, Chaoyi Pan, Nikolai Matni, and Max Simchowitz. Action chunking and data augmentation yield exponential improvements in behavior cloning for continuous spaces. In *The Fourteenth International Conference on Learning Representations*, 2026.

Thomas TCK Zhang, Leonardo Felipe Toso, James Anderson, and Nikolai Matni. Sample-efficient linear representation learning from non-IID non-isotropic data. In *International Conference on Learning Representations*, 2024.

Thomas Tck Zhang, Behrad Moniri, Ansh Nagwekar, Faraz Rahman, Anton Xue, Hamed Hassani, and Nikolai Matni. On the concurrence of layer-wise preconditioning methods and provable feature learning. In *International Conference on Machine Learning*, pp. 75793–75833. PMLR, 2025a.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient LLM training by gradient low-rank projection. In *Forty-first International Conference on Machine Learning*, 2024.

Yang Zhao, Hao Zhang, and Xiuyuan Hu. Penalizing gradient norm for efficiently improving generalization in deep learning. In *International conference on machine learning*, pp. 26982–26992. PMLR, 2022.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. *Advances in neural information processing systems*, 33:18795–18806, 2020.

Liu Ziyin, Zhikang T Wang, and Masahito Ueda. Laprop: Separating momentum and adaptivity in adam. *arXiv preprint arXiv:2002.04839*, 2020.

CONTENTS

1 Introduction **1**

2 Learning under Test-Time Feedback **2**

3 Distribution Shift under Test-Time Feedback **3**

3.1 TTF Shift and Imperfect Feature Learning 4

4 Double Preconditioning (DoPr) **5**

4.1 Theoretical Motivation: AP Mitigates TTF via Isotropic Feature Learning 6

4.2 Lifting AP to General Architectures: An Invariance Principle 6

4.3 Gradient-Preconditioning and Hyperparameter Scaling for DoPr 7

5 Capabilities **7**

5.1 Setting the Stage: Drop-in for Continuous Control 8

5.2 Image-Based Robot Policy Learning 8

5.3 Language Models 8

6 Discussion **9**

A Extended Related Work **20**

B Extended Method Derivation and Mechanisms **23**

B.1 Full DoPr Algorithm 23

B.2 AP For General Architectures: Derivations via Invariance 24

B.3 Approximation Schemes for AP 26

B.3.1 Rank-1 + Diagonal 26

B.3.2 Spatially Uncorrelated Activations (SUA) 26

B.4 Feature Learning Theory 28

B.5 Dynamical Systems Theory 29

B.5.1 Mismatch between $\mathcal{L}_{\text{ideal}}$ and $\mathcal{L}_{\text{val}}$: a construction 30

B.5.2 Proof of Proposition 3.2 33

B.6 Computational Overhead 34

C Experiment Details and Additional Results **35**

C.1 General Operating Guideline 36

C.2 Feature Learning: Validation Loss versus Subspace Distance 36

C.3 Activation Invariance of DoPr 37

C.4 Hyperparameter Scaling DoPr 38

C.5 State-based Imitation Learning 40

C.6 Image-Based Robot Policy Learning 41

C.7 Supervised Fine-Tuning 43
C.7.1 3B SFT 43
C.7.2 8B SFT 45
C.8 Generative Modeling 48

D Additional Experiments and Setup **49**
D.1 Moment Grafting Experiments 49
D.2 Stable Ranks of Optimizer and Model Quantities 51

A Extended Related Work

Imitation learning and the compounding errors problem. The compounding errors problem has been recognized as a fundamental issue in policy learning from demonstration data (Pomerleau, 1988), where small one-step learning errors may shift the learned model away from the offline data distribution it was trained on, leading to potentially catastrophic failure. Traditional approaches to mitigating compounding error come from modifying data collection, either through iterative online collection of recovery behavior (Ross et al., 2011; Kelly et al., 2019; Foster et al., 2024) or noising action execution (Laskey et al., 2017; Zhang et al., 2026) to better regularize the policy geometry locally around the data distribution. More recently, modifying *policy parameterization* has been explored as an intervention for enabling consistent longer-horizon sequential deployment, such as the use of iteratively supervised methods (e.g., diffusion and flow-based parameterizations) (Chi et al., 2023; Pan et al., 2025) and action-chunking (i.e., multi-step prediction and execution) (Shafiullah et al., 2022; Chi et al., 2023; Liu et al., 2025b). Therefore, while data-collection and architecture design have been the primary avenues for mitigating compounding error, comparatively less focus has been directed to the role of the deep learning *optimizer*. An exception of note is Block et al. (2024), which identifies that small loss fluctuations in gradient-based neural network training can cause disproportionately large or even oscillatory performance fluctuations in behavior cloning settings, precisely aligning with our core thesis that small error directions as measured by training loss can be disproportionately harmful when rolled out. Their proposed intervention is a variance-reduction approach via maintaining an EMA-ed copy of the model parameters for evaluation, which is complementary to our preconditioning approach, and as such we adopt in our continuous-space BC experiments. In most other literature, the optimizer is obviated as an oracle yielding (approximate) empirical risk minimizers (see e.g., Ross et al. (2011); Foster et al. (2024); Zhang et al. (2026)). However, the role of specifically the deep learning optimizer has received comparably more attention in deep reinforcement learning, likely due to the compounding difficulties RL presents over supervised learning, such as input- and target- nonstationarity (Lyle et al., 2022; Sokar et al., 2023; Castanyer et al., 2025). In this paper, we expose the potential suboptimalities of a wide range of deep learning optimizers that arise **even in comparatively simpler offline supervised TTF settings**, such as behavior cloning.

Fisher Information, `KFAC`, and prior occurrences of Activation Preconditioning. The Fisher Information matrix is a core object in mathematical statistics. When used as a preconditioner for local optimization in maximum likelihood estimation, the resulting method is known as the *natural gradient method*. From this perspective, the Fisher Information in some cases is related to other curvature matrices such as the Hessian (or the Generalized Gauss-Newton approximation thereof), thus in settings where feasibly implementable, the natural gradient method often greatly accelerates optimization; see e.g., (Amari, 1998; Amari & Nagaoka, 2000; Martens, 2020). Specified to neural networks, which have too many parameters for full second-order methods to be feasible, factorized versions of the Fisher Information have been proposed (Roux et al., 2007; Ollivier, 2013; Pascanu & Bengio, 2013). Most notably, under certain statistical independence assumptions, Martens & Grosse (2015) derive a block-wise *Kronecker-Factored* approximation of the Fisher Information, which renders preconditioning into left- and right- preconditioning the layerwise gradient *matrix* $\mathbf{A}^{-1}\nabla_{\mathbf{W}}\mathcal{L}\mathbf{B}^{-1}$, which spawned the Kronecker-Factored Approximate Curvature (`KFAC`) line of neural network optimizers, seeing consistent development and adaptation (Grosse & Martens, 2016; George et al., 2018; Pauloski et al., 2021; Eschenhagen et al., 2023; Dangel et al., 2025). However,

most literature deriving from KFAC focus on the standard signal of improving train/validation convergence; due to the additional memory overhead of left- and right- preconditioner states per-layer, the numerical nuance of matrix inversion, and relative scarcity rules-of-thumb compared to popular GPs, AdaGrad-based (i.e., GP) methods typically remain favored in mainstream training set-ups, though KFAC finds utility in many scenarios that directly require curvature estimation (He et al., 2025; Ikram et al., 2026). Whereas KFAC involves a "left-side" derivative-based preconditioner, its "right-side" preconditioner is simply the activation covariance, which only requires forward passes to compute. This has motivated "right-side-only" approximations to KFAC (i.e., AP in our parlance) (Benzing, 2022), which have been independently derived from proximal or local layer-wise loss perspectives (Frerix et al., 2018; Amid et al., 2022). We note AP has also been encountered from shallow neural-network learning theory (Zhang et al., 2024), which we discuss in the dedicated feature learning theory paragraph.

Furthermore, a perhaps underutilized perspective of KFAC's approximation of the natural gradient method (or equivalently an alternate derivation of it) is reparameterization invariance; recall that natural gradient *flow* traces a path over distributions that is invariant to smooth reparameterizations. Similarly, KFAC, as an approximation thereof, induces a *layerwise* invariance to affine reparameterizations, see Martens & Grosse (2015, Section 10). The resulting optimizers can be equivalently recovered by a Euclidean-norm steepest descent formulation under a *weighted* local inner product: $\langle \mathbf{G}, \nabla_{\boldsymbol{\theta}} \mathcal{L} \rangle_{\Sigma(\boldsymbol{\theta})}$, s.t. $\|\boldsymbol{\theta}\|_2 \leq \eta$, where $\langle \cdot, \cdot \rangle_{\Sigma(\boldsymbol{\theta})}$ is the classical "Fisher metric" for the natural gradient method, and the self-explanatory "KFAC" metric for KFAC (Luk & Grosse, 2018). Similarly, we may further reduce KFAC to the one-sided AP preconditioning that preserves the invariance property that we argue is salient for TTF in Proposition 4.2, where $\langle \cdot, \cdot \rangle_{\Sigma(\boldsymbol{\theta})}$ is simply the activation-covariance reweighted inner product. Departing from the Euclidean-norm steepest descent to other (layerwise) weight norms recovers DoPr for different GPs (Pethick et al., 2025).

We also note concurrent work in (Du & Su, 2026), which independently derives activation-preconditioning from an "isotropic curvature model" (Su, 2025). In fact, DoPr-Muon in our parlance precisely recovers their proposed algorithm. Crucially, the focus of their work is on improving pretraining loss convergence, while our central thesis studies the benefit of AP independent of loss convergence; notably, we found that DoPr-Muon does not universally accelerate training/validation loss convergence across applications. Overall, we opted to use AP rather than the full KFAC-preconditioned gradient (which would additionally introduce a "left-side" preconditioner) due to subtleties of either requiring an MLE interpretation of the loss function (Martens & Grosse, 2015), contending with possible "incorrectness" of the approximation due to using the Empirical Fisher (Kunstner et al., 2019), and most importantly computing and possibly storing an additional optimizer preconditioner state. Anecdotally, using AP- instead of KFAC- preconditioning in DoPr sufficed to provide the boosts in downstream performance documented in this paper, while avoiding compounding numerical instability or memory boundedness; however, we emphatically do not rule out the possibility for an efficient implementation or adaptation to yield further improvements. Ultimately, we view DoPr as a framework for decoupling *directionality* (e.g., from preconditioning) and *self-normalization* (e.g., from popular GP mechanisms).

Adaptive gradient optimizers in deep learning. Adaptive optimizers have served as the backbone of training deep learning models for more than a decade. In particular, the AdaGrad family of gradient whiteners/normalizers (Duchi et al., 2011) serves as a unifying template for many ensuing optimizers. RMSprop and Adam use a "diagonal" approximation of the AdaGrad gradient covariance matrix for adaptive optimization, while Shampoo (Gupta et al., 2018; Anil et al., 2020; Shi et al., 2023) can be viewed as a layer-wise *Kronecker-Factored* approximation (with one-sided variants (An et al., 2026)), and Muon can be viewed as an instantaneous version of Shampoo, which amounts to (approximately) orthogonalizing layerwise gradient matrices. From there, many further reductions or augmentations have been introduced. To name just a few, within the "Adam family", we have the precursor RMSprop (Tieleman & Hinton, 2012), AdaBelief (Zhuang et al., 2020), LAProp (Ziyin et al., 2020), Lion (Chen et al., 2023), MVN-Grad (Patitucci & Mokhtari, 2026), augmented versions AdEMAMix (Pagliardini et al., 2025), as well as reduced versions AdaFactor (Shazeer & Stern, 2018), GaLore (Zhao et al., 2024), Adam-mini (Zhang et al., 2025b), among many more; see e.g., (Schmidt et al., 2021) for a more comprehensive account. On the matrix-shaped adaptive optimizer side, there have been recent developments on top of Shampoo, e.g., through adaptive optimization in an eigenbasis-adjusted gradient (SOAP (Vyas et al., 2024), SPlus (Frans et al., 2026), KL-Shampoo (Lin et al., 2025)), or additional adaptivity applied to Muon, e.g., AdaMuon

(Si et al., 2025), NorMuon (Li et al., 2025), MuonMax (Crawshaw et al., 2025). We also note the PSGD line of work (Li, 2017; 2018), which spawns distinct preconditioners using various Lie group factorizations. Lastly, we mention a line of work that models (and creates) optimizers based on interpretations as steepest descent with respect to different choices of layerwise norm (Carlson et al., 2015; Bernstein & Newhouse, 2025; Kovalev, 2025; Pethick et al., 2025). We note that, with few exceptions, all the aforementioned optimizers are of a *whitening* or *normalizing* flavor, with the goal of ensuring the update direction is (approximately) well-conditioned in various senses, e.g., entry-wise, row/column-wise, or spectrum-wise. Though there are far too many optimizers in this list whose efficacy as GP's in our DoPr paradigm can be verified, our experiments on representative examples and general hypothesis is that any such "normalizing-type" GP is compatible with the AP gradient.

Neural network feature learning theory. A by now classical approach to understanding how neural networks learn is via the neural tangent kernel (NTK) (Jacot et al., 2018; Mei & Montanari, 2022). However, kernel methods rely on fixed, features, which fundamentally limits their expressivity and leads to suboptimal sample complexity for learning nonlinear functions (Ghorbani et al., 2021a;b). This has motivated a large body of subsequent work studying feature learning in neural networks. In the setting of **isotropic covariates**, it has been shown that even a single step of SGD on the first layer of a two-layer network can learn sufficiently informative features to achieve improved sample complexity over kernel methods, provided the target function has low-dimensional structure (Damian et al., 2022; Moniri et al., 2024). In contrast, for general anisotropic covariates, Zhang et al. (2024) and Zhang et al. (2025a) identify that SGD is heavily distorted by anistropy of the inputs, which can cause the subspace distance between the learned and ground-truth representations to grow, even as training loss decreases. They reconcile this issue with layerwise preconditioning schemes; the proposed algorithm in Zhang et al. (2024) is a two-layer version of pure AP (or equivalently DoPr-GD).

Hyperparameter scaling conditions. A complementary line of work studies how optimizer hyperparameters should scale with model width, depth, and parameterization to ensure stable training and hyperparameter transfer. The Tensor Programs framework (Yang, 2019; 2020) provides a unifying theory of infinite-width limits, and maximal update parameterization (μP) (Yang & Hu, 2021; Yang et al., 2022) derives learning-rate and initialization scalings that keep activations $\Theta(1)$ while preserving maximal feature learning across widths. Subsequent work extends these principles to second-order and matrix-preconditioned optimizers such as KFAC and Shampoo (Ishikawa & Karakida, 2023), alternative formulations based on spectral norms (Yang et al., 2023), depth-wise scaling laws and residual multipliers (Dey et al., 2025; Yang et al., 2024), and newer optimizers including SOAP and Muon (Qiu et al., 2025). Recent work further shows that for long training runs, weight decay scaling can dominate stability, with some works advocating constant learning-rate–weight-decay products (Wang & Aitchison, 2025; Kosson et al., 2025) and width-dependent weight decay scaling (Qiu et al., 2025). Broadly, these methods derive optimizer-specific scaling rules, often through single-step update analyses, to enable efficient hyperparameter tuning across model sizes. Our work enables painless integration of these findings: by layering the GPs studied in these works on top of the AP update, we are able to make immediate use of these scaling rules.

Mismatch between training performance and downstream behavior. Beyond anecdotal or folklore understanding that training/validation loss across optimizers or even hyperparameters need not correlate to better downstream performance, there have been various works trying to understand or ameliorate this phenomenon. From the optimization perspective, recent work has identified that increasing pre-training effort (e.g., duration or tokens)—thus improving training performance—may lead to worse downstream finetunability (Springer et al., 2025; Horoi et al., 2025). On the other hand, works since Chinchilla (Hoffmann et al., 2022) have steadily supported that pretraining token-per-parameter scaling laws can be in practice be much larger (Gadre et al., 2025) while downstream performance improves; notably, this includes prominent open-weight models such as Llama 3 (Meta, 2024). Beyond coarse optimization interventions such as data quantity and training duration, certain works in the context of LLM fine-tuning have identified suboptimality in downstream performance when using different optimizers during pretraining and finetuning, e.g., Adam v.s. Muon, notably documented in Moonshot's whitepaper introducing large-scale Muon-trained models (Liu et al., 2025a), as well as follow-up studies (Liu et al., 2026c). We note that our fine-tuning experiments use open-weight pre-trained models, and thus using DoPr for finetuning departs from the optimizer-consistency paradigm—it remains to verify: 1. if pretraining with DoPr

can further extend its benefits, 2. if our observed benefits using DoPr for finetuning hold at scale and for general architectures. In a related perspective to ours regarding the benefits of "uniform" feature learning, some works have proposed using Sharpness-Aware Minimization (SAM) and extensions thereof as an *optimization* intervention to yield better zero-shot or few-shot performance of the model on downstream tasks (Foret et al., 2021; Bahri et al., 2022; Zhao et al., 2022), even at the cost of (pre-)training validation loss (Watts et al., 2026). However, the mechanisms behind when SAM can yield reliable benefits as well as the feature geometry it actually induces do not necessarily reflect the nominal purpose of SAM (Andriushchenko et al., 2023; Schapiro & Zhao, 2024).

As a parallel line of literature, many works have explored proxy statistics or metrics beyond naive held-out/validation loss on the training objective toward predicting downstream performance, robustness, or adaptability. As a small sample, we highlight certain works that explore *jointly* accounting for gradient- and activation-statistics (Liu et al., 2026b;a). In Liu et al. (2026b), a gradient-weighted activation statistic is proposed in the context of detecting plasticity loss in RL networks, a phenomenon by which much of a network's capacity is silently wasted; notably, in RL contexts the training objective is nonstationary, and thus instantaneous training losses are often less predictive than in supervised learning. As our work focuses on supervised contexts, we leave investigating DoPr's effectiveness in RL contexts for future work. Lastly, we note evidence cautioning using internal network statistics for concluding downstream performance; e.g., recent work finds that certain network trends typically associated with better/worse downstream performance may rather reflect current training practices, rather than a causal relationship with performance (Kulkarni et al., 2026; Liu et al., 2026a).

B Extended Method Derivation and Mechanisms

B.1 Full DoPr Algorithm

In this section, we list the full DoPr algorithm and draw attention to various important design decisions we made to obtain performant results.

Algorithm 2 Double Preconditioning (Feedforward Layer, layer index ℓ suppressed)

Input: Base Optimizer GP,learning rate η, weight decay λ, damping γ, EMA rate β
for $k = 1$ **to** K **do**
  Sample batch B, $|B| = n$
  $\mathbf{G} \leftarrow \nabla_{\mathbf{W}} \widehat{\mathcal{L}}(\boldsymbol{\theta}^{(k-1)})$
  $\Sigma \leftarrow \frac{1}{n} \sum_{i=1}^{n} \mathbf{z}_i^{(k-1)} \mathbf{z}_i^{(k-1)\top}$
  $\overline{\mathbf{G}}^{(k)} \leftarrow \beta \overline{\mathbf{G}}^{(k-1)} + (1-\beta)\mathbf{G}$
  $\overline{\Sigma}^{(k)} \leftarrow \beta \overline{\Sigma}^{(k-1)} + (1-\beta)\Sigma$
  $\mathbf{M}^{(k)} \leftarrow \overline{\mathbf{G}}^{(k)} \big(\overline{\Sigma}^{(k)} + \gamma \operatorname{tr}(\overline{\Sigma}^{(k)})\mathbf{I}\big)^{-1}$ (AP)
  $\mathbf{D}^{(k)} \leftarrow \mathrm{GP}(\mathbf{M}^{(k)}, {}^{**}\mathrm{kwargs})$ (DoPr)
  $\mathbf{W}^{(k)} \leftarrow (1-\eta\lambda)\mathbf{W}^{(k-1)} - \eta \mathbf{D}^{(k)}$
end for

Handling Different Layer Types While we define AP for linear layers, we extend it to other common architectural primitives by reducing them to equivalent linear maps. We unfold Conv1d layers and treat them akin to linear layers. For Conv2d layers, see Appendix B.2, and for Attention Layers, see Appendix B.2. Finally, for affine conditioning modules such as FiLM (Perez et al., 2018) and AdaLN (Peebles & Xie, 2023), we recommend first trying DoPr, as the performance of these modules can be sensitive to the relative learning speeds of the network backbone (which is presumably being trained with DoPr). We note that zero-initialized networks (e.g., in AdaLN-Zero) may require slight attention as initial activations may be zero, potentially destabilizing early training due to ill-conditioned AP. We found that with damping and the self-normalizing property of popular GPs, DoPr can work without modification, but if early training is unstable, we recommend a short warmup period with the base GP before switching to the full DoPr.

Decoupled Weight Decay This stabilizes training by controlling parameter norms independently of the gradient update (Loshchilov, 2017).

EMA We provide the option to maintain exponential moving averages of both the gradient and the activation covariance. For a generic estimator $\mathbf{G}^{(k)}$, the EMA update is

$$\bar{\mathbf{G}}^{(k)} = \beta\bar{\mathbf{G}}^{(k-1)} + (1-\beta)\mathbf{G}^{(k)}.$$

Unrolling the recursion gives

$$\bar{\mathbf{G}}^{(k)} = (1-\beta)\sum_{t=0}^{k-1}\beta^t\mathbf{G}^{(k-t)} + \beta^k\bar{\mathbf{G}}^{(0)},$$

i.e., a geometrically weighted average of past iterates. Assuming $\mathbf{G}^{(k)}$ are unbiased estimators of some quantity μ with covariance Σ_G, we have

$$\mathbb{E}[\bar{\mathbf{G}}^{(k)}] = (1-\beta^k)\mu + \beta^k\bar{\mathbf{G}}^{(0)},$$

so the estimator is biased at early iterations, but converges to μ at rate β^k. For the variance, assuming independence for simplicity,

$$\mathrm{Var}(\bar{\mathbf{G}}^{(k)}) = (1-\beta)^2\sum_{t=0}^{k-1}\beta^{2t}\Sigma_G \;\longrightarrow\; \frac{1-\beta}{1+\beta}\Sigma_G \quad \text{as } k\to\infty.$$

Thus EMA reduces variance by a factor of $\frac{1-\beta}{1+\beta}$ while introducing a bias, which becomes negligible for sufficiently large k. We provide the option of applying EMA to both the gradient and the activation covariance. **We recommend first trying `AP` in `DoPr` *without* additional EMAs:** 1. most of our experiments compute `AP` with just the batch-wise gradient and activation covariance, 2. many base `GP`s are equipped with their own gradient EMAs (e.g. `Adam` and `Muon`), and thus applying gradient EMA in the `AP` computation requires accounting for doubly applying EMA, 3. using batch-wise statistics in `AP` removes the additional persistent memory requirement for the activation covariance buffer, making `DoPr`'s memory cost equal to its base `GP`.

Damping We add a damping term to the activation covariance estimate of the form $\Sigma_{\mathbf{z}} + \lambda\frac{\mathrm{tr}(\Sigma_{\mathbf{z}})}{\mathrm{d}_{\mathbf{z}}}\mathbf{I}$ to ensure invertibility. We note that the damping is set *relative* to the magnitude of $\Sigma_{\mathbf{z}}$ via $\frac{\mathrm{tr}(\Sigma_{\mathbf{z}})}{\mathrm{d}_{\mathbf{z}}}$ such that the damping hyperparameter is scale-invariant (Ishikawa & Karakida, 2023; Dey et al., 2025). Though damping the activation covariance has conceptual connections with ridge regression, we posit that damping, and indeed using generic PD linsolvers that requires full-rankness such as Cholesky, is likely crude, as damping causes small activation directions (which `AP` aims to upweight) to be boosted equally as *zero* directions (which `AP` should in principle keep zero). We leave designing flexible, fast linsolvers that better handle rank degeneracy and pseudo-inversion (e.g., extensions of the lauded Newton-Schulz iteration in `Muon`) to future work.

Inversion and Approximations For most layers, we compute the inverse of the activation covariance using PyTorch's implementation of Cholesky decomposition `torch.linalg.cholesky_ex`. In our experiments, this was tractable for the considered model sizes, and usually not the compute bottleneck. This is in line with empirical insight that matrix operations that scale independently of the batch size (such as in computing $\mathbf{G}\Sigma^{-1}$) become negligible in large-batch training. For certain settings with inflated activation dimensions, such as in the im2col parameterization of convolutional layers, we also support structured approximations such as rank-1+diagonal (derived in Appendix B.3.1) and rank-k plus diagonal decompositions (Ahn et al., 2025a). For convolutional layers specifically, in line with `KFAC`'s development, we employ the Spatially Uncorrelated Approximation (SUA) Appendix B.3.2 to bypass the (mostly redundant) cost of forming and inverting large patch covariance matrices. We also remark that for LLM embedding layers lead to input (i.e., "activation") dimensions that scale with comparatively huge vocabulary size: however, we note that due to the one-hot nature of the inputs, the activation covariance is diagonal, where each non-zero entry is just the count of that token in the batch, and thus `AP`-ing the embedding layer gradient can be performed extremely efficiently and does not require forming a full $\mathrm{d}_{\mathrm{vocab}} \times \mathrm{d}_{\mathrm{vocab}}$ matrix.

B.2 `AP` For General Architectures: Derivations via Invariance

This section contains proofs for invariance under non-degenerate layer-wise affine transforms for feedforward, convolutional, and attention layers.

Feedforward Affine Invariance

Proposition 4.2. *Consider a feedforward network, and consider the weights and activations* $\{(\mathbf{W}_\ell, \mathbf{z}_\ell)\}_{\ell\in[L]}$ *under non-degenerate layer-wise affine transforms defined in* (4.2)*. Then, given the following layer-wise update rule:* $\mathbf{W}^{\text{next}} = \mathbf{W} - \eta\nabla_{\mathbf{W}}\mathcal{L}(f_{\boldsymbol{\theta}})\boldsymbol{\Sigma}_{\mathbf{z}}^{-1}$, $\boldsymbol{\Sigma}_{\mathbf{z}} = \mathbb{E}[\mathbf{z}\mathbf{z}^\top]$ (AP)*, the updated weights satisfy* $\mathbf{W}_\ell^{\text{next}}\mathbf{z}_\ell = \overline{\mathbf{W}}_\ell^{\text{next}}\overline{\mathbf{z}}_\ell$ *for all* $\ell \in [L]$, $\mathbf{z}_1 = \mathbf{x}, \overline{\mathbf{z}}_1 = \mathbf{A}_1\mathbf{x}$.

Proof. Recall that:

$$\begin{aligned}\boldsymbol{\Sigma}_{\mathbf{z}}^{-1} &= \left(\mathbb{E}\left[\overline{\mathbf{z}}_\ell\overline{\mathbf{z}}_\ell^\top\right]\right)^{-1} \\ &= \left(\mathbb{E}\left[\mathbf{A}_\ell\mathbf{z}_\ell\mathbf{z}_\ell^\top\mathbf{A}_\ell^\top\right]\right)^{-1} \\ &= \left(\mathbf{A}_\ell^\top\right)^{-1}\boldsymbol{\Sigma}_\ell^{-1}\mathbf{A}_\ell^{-1}.\end{aligned}$$

By the chain rule

$$\begin{aligned}\nabla_{\overline{\mathbf{W}}_\ell}\mathcal{L}\left(f_{\bar{\theta}}\right) &= \mathbb{E}\left[\frac{\partial\mathcal{L}}{\partial h}\overline{\mathbf{z}}_\ell^\top\right] \\ &= \mathbb{E}\left[\frac{\partial\mathcal{L}}{\partial h}\mathbf{z}_\ell^\top\right]\mathbf{A}_\ell^\top.\end{aligned}$$

Therefore:

$$\begin{aligned}\overline{\mathbf{W}}_\ell^{\text{next}}\overline{\mathbf{z}}_\ell &= \left(\overline{\mathbf{W}}_\ell - \eta\nabla_{\overline{\mathbf{W}}_\ell}\mathcal{L}\left(f_{\bar{\theta}}\right)\boldsymbol{\Sigma}_{\overline{\mathbf{z}}_\ell}^{-1}\right)\overline{\mathbf{z}}_\ell \\ &= \left(\mathbf{W}_\ell\mathbf{A}_\ell^{-1} - \eta\mathbb{E}\left[\frac{\partial\mathcal{L}}{\partial h}\mathbf{z}_\ell^\top\right]\mathbf{A}_\ell^\top\left(\mathbf{A}_\ell^\top\right)^{-1}\boldsymbol{\Sigma}_{\mathbf{z}_\ell}^{-1}\mathbf{A}_\ell^{-1}\right)\mathbf{A}_\ell\mathbf{z}_\ell \\ &= \left(\mathbf{W}_\ell - \eta\mathbb{E}\left[\frac{\partial\mathcal{L}}{\partial h}\mathbf{z}_\ell^\top\right]\boldsymbol{\Sigma}_{\mathbf{z}_\ell}^{-1}\right)\mathbf{z}_\ell \\ &= \mathbf{W}_\ell^{\text{next}}\mathbf{z}_\ell\end{aligned}$$

Since the argument holds for arbitrary ℓ, the conclusion holds for all layers, and thus the network output.

□

Convolutional Affine Invariance

A convolutional layer can be written as a linear map applied to unfolded input patches via the standard *im2col*parameterization. Specifically, let $\mathbf{Z}_\ell = \text{im2col}(\mathbf{z}_\ell) \in \mathbb{R}^{d_{\text{patch}}\times N}$ denote the matrix of flattened patches, where each column $\mathbf{z}_{\ell,b,p}$ corresponds to a spatial patch indexed by batch element b and location p, and let $\mathbf{W}_\ell \in \mathbb{R}^{C_{\text{out}}\times d_{\text{patch}}}$ denote the flattened convolutional kernel. Then the layer can be written as $\mathbf{H}_\ell = \mathbf{W}_\ell\mathbf{Z}_\ell$.

Corollary B.1 (Convolutional Affine invariance). *Consider a convolutional layer in im2col form with patch matrix* $\mathbf{Z}_\ell$ *and weights* $\mathbf{W}_\ell$*, and consider the transformed variables* $\overline{\mathbf{Z}}_\ell = \mathbf{A}_\ell\mathbf{Z}_\ell$, $\overline{\mathbf{W}}_\ell = \mathbf{W}_\ell\mathbf{A}_\ell^{-1}$ *for invertible* $\mathbf{A}_\ell$*. Then under the AP update* $\mathbf{W}_\ell^{\text{next}} = \mathbf{W}_\ell - \eta\nabla_{\mathbf{W}_\ell}\mathcal{L}(f_\theta)\boldsymbol{\Sigma}_{\mathbf{Z}_\ell}^{-1}$ *with* $\boldsymbol{\Sigma}_{\mathbf{Z}_\ell} = \mathbb{E}_{b,p}[\mathbf{z}_{\ell,b,p}\mathbf{z}_{\ell,b,p}^\top]$*, the updated weights satisfy* $\overline{\mathbf{W}}_\ell^{\text{next}}\overline{\mathbf{Z}}_\ell = \mathbf{W}_\ell^{\text{next}}\mathbf{Z}_\ell$.

Proof. Under the im2col representation, the convolutional layer is a linear map applied to patch vectors. The transformation $\overline{\mathbf{Z}}_\ell = \mathbf{A}_\ell\mathbf{Z}_\ell$, $\overline{\mathbf{W}}_\ell = \mathbf{W}_\ell\mathbf{A}_\ell^{-1}$ preserves pre-activations, i.e. $\overline{\mathbf{W}}_\ell\overline{\mathbf{Z}}_\ell = \mathbf{W}_\ell\mathbf{Z}_\ell$. Thus the setting reduces exactly to the feedforward case of Proposition 4.3 applied to the patch vectors $\mathbf{z}_{\ell,b,p}$, yielding the claim. □

Attention Affine Invariance

A self-attention layer first applies learned linear projections to an input token matrix $\mathbf{Z}_\ell \in \mathbb{R}^{d_{\text{model}}\times N}$:

$$\begin{aligned}\mathbf{Q}_\ell &= \mathbf{W}_\ell^Q\mathbf{Z}_\ell, \\ \mathbf{K}_\ell &= \mathbf{W}_\ell^K\mathbf{Z}_\ell, \\ \mathbf{V}_\ell &= \mathbf{W}_\ell^V\mathbf{Z}_\ell.\end{aligned}$$

The attention output is then computed as a deterministic function of these projected activations, for example

$$\mathrm{Attn}(\mathbf{Q}_\ell, \mathbf{K}_\ell, \mathbf{V}_\ell) = \mathbf{V}_\ell \,\mathrm{softmax}\left(\mathbf{K}_\ell^\top \mathbf{Q}_\ell / \sqrt{d}\right),$$

up to convention-dependent transposes. Thus, if an affine change of coordinates is applied to the input token representation and the inverse transformation is absorbed into each projection matrix, the projected queries, keys, and values are unchanged.

Corollary B.2 (Attention projection AP invariance). *Consider a self-attention layer with input activations $\mathbf{Z}_\ell$ and projection weights $\{\mathbf{W}_\ell^r\}_{r\in\{Q,K,V\}}$. Let $\mathbf{A}_\ell$ be invertible, and define $\overline{\mathbf{Z}}_\ell = \mathbf{A}_\ell \mathbf{Z}_\ell$ and $\overline{\mathbf{W}}_\ell^r = \mathbf{W}_\ell^r \mathbf{A}_\ell^{-1}$ for each $r \in \{Q, K, V\}$. Then under the AP update $(\mathbf{W}_\ell^r)^{\mathrm{next}} = \mathbf{W}_\ell^r - \eta \nabla_{\mathbf{W}_\ell^r} \mathcal{L}(f_\theta) \mathbf{\Sigma}_{\mathbf{Z}_\ell}^{-1}$, where $\mathbf{\Sigma}_{\mathbf{Z}_\ell} = \mathbb{E}_{b,t}[\mathbf{z}_{\ell,b,t}\mathbf{z}_{\ell,b,t}^\top]$, the updated projections satisfy $(\overline{\mathbf{W}}_\ell^r)^{\mathrm{next}} \overline{\mathbf{Z}}_\ell = (\mathbf{W}_\ell^r)^{\mathrm{next}} \mathbf{Z}_\ell$ for each $r \in \{Q, K, V\}$. Consequently, the updated attention queries, keys, values, and attention outputs are invariant under the transformed parameterization.*

Proof. For each projection $r \in \{Q, K, V\}$, the map $\mathbf{Z}_\ell \mapsto \mathbf{W}_\ell^r \mathbf{Z}_\ell$ is a feedforward linear layer applied to token activations. Moreover, the transformed parameterization preserves the corresponding projected activations:

$$\begin{aligned} \overline{\mathbf{W}}_\ell^r \overline{\mathbf{Z}}_\ell &= \mathbf{W}_\ell^r \mathbf{A}_\ell^{-1} \mathbf{A}_\ell \mathbf{Z}_\ell \\ &= \mathbf{W}_\ell^r \mathbf{Z}_\ell. \end{aligned}$$

Therefore Proposition 4.3 applies separately to the query, key, and value projections, yielding

$$(\overline{\mathbf{W}}_\ell^r)^{\mathrm{next}} \overline{\mathbf{Z}}_\ell = (\mathbf{W}_\ell^r)^{\mathrm{next}} \mathbf{Z}_\ell \qquad \text{for each } r \in \{Q, K, V\}.$$

Thus the updated queries, keys, and values are identical in the two parameterizations. Since the attention output is a deterministic function of $\mathbf{Q}_\ell, \mathbf{K}_\ell, \mathbf{V}_\ell$, the updated attention output is also identical. □

B.3 Approximation Schemes for AP

For larger models, particularly those with 2d convolutions, storing and computing the full activation covariance used in the preconditioner can be expensive. To subvert this cost, we develop multiple preconditioning approximation schemes.

B.3.1 Rank-1 + Diagonal

For a random vector $\mathbf{x}$ supported on $\mathbb{R}^d$, let $\mu = \mathbb{E}[\mathbf{x}]$ and $\mathrm{Var} = \mathbb{E}\left[\mathbf{x}\mathbf{x}^\top\right] - \mu\mu^\top \succeq \mathbf{0}$. Therefore, we have the second moment satisfies $\Sigma = \mathbb{E}[\mathbf{x}\mathbf{x}^\top] = \mu\mu^\top + \mathrm{Var}$. By approximating Var with its diagonal, we find a rank-1 + diagonal approximation:

$$\Sigma + \lambda \mathbf{I} \approx \mathrm{diag}\,(\mathrm{Var}) + \lambda \mathbf{I} + \mu\mu^\top$$

Let $\mathbf{D} = \mathrm{diag}\,(\mathrm{Var}) + \lambda I$. Since $\mathbf{D}$ is diagonal and $\mu\mu^\top$ is rank-1, by the Sherman-Morrison formula we may compute

$$(\Sigma + \lambda \mathbf{I})^{-1} \approx \left(\mathbf{D} + \mu\mu^\top\right)^{-1} \tag{B.1}$$

$$= \mathbf{D}^{-1} - \frac{\left(\mathbf{D}^{-1}\mu\mu^\top\mathbf{D}^{-1}\right)}{1 + \mu^\top \mathbf{D}^{-1}\mu}, \tag{B.2}$$

which can be computed in linear time with respect to the hidden dimension d, and only requires persistent storage of $\mu, \mathbf{D}$ scaling linear in d if performing EMA on the preconditioner.

B.3.2 Spatially Uncorrelated Activations (SUA)

We derive the SUA (George et al., 2018) approximation for a single 2D convolutional layer using an `im2col` parameterization.

Let the (tensor-shaped) convolutional weights be $\mathbf{W} \in \mathbb{R}^{C_{out} \times C_{in} \times k_h \times k_w}$ and let $\boldsymbol{\theta} = \mathrm{vec}(\mathbf{W})$. For each example $i \in [n]$ and output spatial index $u \in [m]$ (where $m = H_{out}W_{out}$), let $\mathbf{a}_{i,u} \in \mathbb{R}^d$ denote the vectorized input patch (concatenating channels and kernel offsets) with

$$d \;=\; C_{in}k_hk_w,$$

and let $\mathbf{g}_{i,u} \in \mathbb{R}^{C_{out}}$ denote the gradient of the loss w.r.t. the pre-activation at location u. Define the stacked matrices

$$\mathbf{A}_i \triangleq [\mathbf{a}_{i,1} \quad \cdots \quad \mathbf{a}_{i,m}]^\top \in \mathbb{R}^{m \times d}, \qquad \mathbf{G}_i \triangleq [\mathbf{g}_{i,1} \quad \cdots \quad \mathbf{g}_{i,m}]^\top \in \mathbb{R}^{m \times C_{out}}.$$

Then the per-example weight gradient (reshaped to $C_{out} \times d$) is

$$\nabla_{\mathbf{W}} \ell(\mathbf{x}_i) \;=\; \mathbf{G}_i^\top \mathbf{A}_i \;\in\; \mathbb{R}^{C_{out} \times d}.$$

Using $\mathrm{vec}(\mathbf{XYZ}) = (\mathbf{Z}^\top \otimes \mathbf{X})\mathrm{vec}(\mathbf{Y})$ gives

$$\nabla_{\boldsymbol{\theta}} \ell(\mathbf{x}_i) \;\triangleq\; \mathrm{vec}(\nabla_{\mathbf{W}} \ell(\mathbf{x}_i)) \;=\; \mathrm{vec}(\mathbf{G}_i^\top \mathbf{A}_i) \;=\; (\mathbf{A}_i^\top \otimes \mathbf{I}_{C_{out}})\,\mathrm{vec}(\mathbf{G}_i^\top).$$

The empirical Fisher for this layer is

$$\hat{\mathbf{F}} \;\triangleq\; \widehat{\mathbb{E}}\Big[\nabla_{\boldsymbol{\theta}} \ell(\mathbf{x})\, \nabla_{\boldsymbol{\theta}} \ell(\mathbf{x})^\top\Big] \;=\; \frac{1}{n} \sum_{i=1}^{n} \sum_{u=1}^{m} \sum_{v=1}^{m} \Big(\mathbf{a}_{i,u}\mathbf{a}_{i,v}^\top\Big) \otimes \Big(\mathbf{g}_{i,u}\mathbf{g}_{i,v}^\top\Big).$$

The double sum over (u, v) shows that the exact curvature couples *pairs of spatial locations*, making exact storage/inversion expensive when m is large.

KFAC Grosse & Martens (2016) applies the standard approximation that (patch) activations and backpropagated gradients are independent:

$$\mathbb{E}\Big[(\mathbf{a}_u\mathbf{a}_v^\top) \otimes (\mathbf{g}_u\mathbf{g}_v^\top)\Big] \;\approx\; \mathbb{E}[\mathbf{a}_u\mathbf{a}_v^\top] \;\otimes\; \mathbb{E}[\mathbf{g}_u\mathbf{g}_v^\top].$$

This reduces the Fisher to products of second moments of $\mathbf{a}$ and $\mathbf{g}$.

SUA further assumes spatial uncorrelatedness of activations:

$$\mathbb{E}[\mathbf{a}_u\mathbf{a}_v^\top] \;=\; \mathbf{0} \qquad \text{for } u \neq v,$$

(and analogously, in many implementations, $\mathbb{E}[\mathbf{g}_u\mathbf{g}_v^\top] \approx \mathbf{0}$ for $u \neq v$). Under this approximation, only the diagonal terms $u = v$ remain, yielding

$$\mathbf{F} \;\approx\; \sum_{u=1}^{m} \mathbb{E}[\mathbf{a}_u\mathbf{a}_u^\top] \;\otimes\; \mathbb{E}[\mathbf{g}_u\mathbf{g}_u^\top].$$

If we additionally assume stationarity across spatial locations (i.e. the second moments do not depend on u), then

$$\Sigma_{\mathbf{a}} \;\triangleq\; \mathbb{E}[\mathbf{a}_u\mathbf{a}_u^\top] \in \mathbb{R}^{d \times d}, \qquad \Sigma_{\mathbf{g}} \;\triangleq\; \mathbb{E}[\mathbf{g}_u\mathbf{g}_u^\top] \in \mathbb{R}^{C_{out} \times C_{out}},$$

and thus

$$\mathbf{F} \;\approx\; m\,(\Sigma_{\mathbf{a}} \otimes \Sigma_{\mathbf{g}}),$$

where the scalar factor m is absorbed into the step size / damping.

We estimate the factors by averaging over both batch and spatial indices:

$$\hat{\Sigma}_{\mathbf{a}} \;=\; \frac{1}{nm} \sum_{i=1}^{n} \sum_{u=1}^{m} \mathbf{a}_{i,u}\mathbf{a}_{i,u}^\top \;=\; \widehat{\mathbb{E}}_{i,u}[\mathbf{a}_{i,u}\mathbf{a}_{i,u}^\top], \qquad \hat{\Sigma}_{\mathbf{g}} \;=\; \frac{1}{nm} \sum_{i=1}^{n} \sum_{u=1}^{m} \mathbf{g}_{i,u}\mathbf{g}_{i,u}^\top \;=\; \widehat{\mathbb{E}}_{i,u}[\mathbf{g}_{i,u}\mathbf{g}_{i,u}^\top].$$

Notably, $\hat{\Sigma}_{\mathbf{a}}$ has size $(C_{in}k_hk_w) \times (C_{in}k_hk_w)$ and is independent of the feature-map resolution or batch size (and reduces to $C_{in} \times C_{in}$ for 1×1 convolutions).

Let $\mathbf{g}_W \triangleq \nabla_{\mathbf{W}} \ell$ be the weight gradient in matrix form $\mathbb{R}^{C_{out} \times d}$. Under the Kronecker-factored approximation $\mathbf{F} \approx \Sigma_{\mathbf{a}} \otimes \Sigma_{\mathbf{g}}$, the (damped) KFAC/SUA preconditioner applies

$$\Delta\mathbf{W} \;\propto\; (\Sigma_{\mathbf{g}} + \lambda\mathbf{I})^{-1}\, \mathbf{g}_W\, (\Sigma_{\mathbf{a}} + \lambda\mathbf{I})^{-1},$$

followed by reshaping $\Delta\mathbf{W}$ back to the tensor shape of $\mathbf{W}$. Importantly, SUA does not drastically reduce the expressivity of the preconditioner as the discarded curvature directions correspond to spatial interaction modes that are unidentifiable under weight sharing. As such, they lie largely in the nullspace of the parameterization. Note that we only use the right-sided SUA.

B.4 Feature Learning Theory

We adapt parts of Zhang et al. (2024) and Zhang et al. (2025a) toward establishing the feature learning result in Proposition 4.1. We note for presentational convenience, we make some technical assumptions:

Assumption B.1. The current iterate $\mathbf{G}_\theta$ satisfies:

1. $\mathbf{G}_\theta \in \mathbb{R}^{d\times n}$ is row-orthonormal.
2. $\mathrm{dist}(\mathbf{G}_\theta, \mathbf{G}_\star) \le \frac{\lambda_{\min}(\mathbf{F}_\star{}^\top \mathbf{F}_\star)\,\lambda_{\min}(\mathbf{\Sigma_s})}{\lambda_{\max}(\mathbf{F}_\star{}^\top \mathbf{F}_\star)\,\lambda_{\max}(\mathbf{\Sigma_s})}$.

Proposition B.1 (Proposition 4.1, formal ver.). *Let Assumption B.1 hold. Given* $0 < \eta \le 0.5\,\lambda_{\max}(\mathbf{F}_\star{}^\top \mathbf{F}_\star)^{-1}$. *Then, one step of full-batch* GD *(resp.* AP*) satisfies:*

$$\begin{aligned}\mathbf{G}_\theta^{\mathrm{next,GD}}\mathcal{P}_\star^\perp &= (\mathbf{G}_\theta - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta \mathbf{G}_\theta \mathbf{\Sigma_s})\mathcal{P}_\star^\perp + \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\star \mathbf{G}_\star \mathbf{\Sigma_s}\mathcal{P}_\star^\perp \\ \mathbf{G}_\theta^{\mathrm{next,AP}}\mathcal{P}_\star^\perp &= (\mathbf{I} - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta)\mathbf{G}_\theta \mathcal{P}_\star^\perp.\end{aligned} \tag{B.3}$$

Then, the AP *update satisfies the following one-step contraction in subspace distance:* $\mathrm{dist}(\mathbf{G}_\theta^{\mathrm{next,AP}}, \mathbf{G}_\star) \le \big(1 - \eta\,\lambda_{\min}(\mathbf{F}_\theta{}^\top \mathbf{F}_\theta)\big)\,\mathrm{dist}(\mathbf{G}_\theta, \mathbf{G}_\star)$. *On the other hand, one may construct* $\mathbf{F}_\theta$, $\mathbf{F}_\star$, $\mathbf{G}_\star$, $\mathbf{\Sigma_s}$ *such that* $\mathrm{dist}(\mathbf{G}_\theta^{\mathrm{next,GD}}, \mathbf{G}_\star) > \mathrm{dist}(\mathbf{G}_\theta, \mathbf{G}_\star)$.

Proof of Proposition 4.1. First, recalling the loss: $\mathcal{L}(\mathbf{F}_\theta \mathbf{G}_\theta) = \mathbb{E}^{\pi_{\mathrm{demo}}}\|\mathbf{F}_\theta \mathbf{G}_\theta - \mathbf{s}_t\|_2^2 = \|(\mathbf{F}_\theta \mathbf{G}_\theta - \mathbf{F}_\star \mathbf{G}_\star)\mathbf{\Sigma_s}^{1/2}\|_F^2$, we have the GD and AP update:

$$\begin{aligned}\mathbf{G}_\theta^{\mathrm{next,GD}} &= \mathbf{G}_\theta - \eta \nabla_{\mathbf{G}_\theta}\mathcal{L} \\ &= \mathbf{G}_\theta - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta \mathbf{G}_\theta \mathbf{\Sigma_s} + \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\star \mathbf{G}_\star \mathbf{\Sigma_s} \\ \mathbf{G}_\theta^{\mathrm{next,AP}} &= \mathbf{G}_\theta - \eta \nabla_{\mathbf{G}_\theta}\mathcal{L} \cdot \mathbf{\Sigma_s}^{-1} \\ &= \mathbf{G}_\theta - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta \mathbf{G}_\theta + \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\star \mathbf{G}_\star.\end{aligned}$$

□

Noting $\mathbf{G}_\star \mathcal{P}_\star^\perp = \mathbf{0}$, applying $\mathcal{P}_\star^\perp$ to the above yields (4.1). Now, given that

$$\mathbf{G}_\theta^{\mathrm{next,AP}}\mathcal{P}_\star^\perp = (\mathbf{I} - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta)\mathbf{G}_\theta \mathcal{P}_\star^\perp,$$

applying $\|\cdot\|_{\mathrm{op}}$ and submultiplicativity on the right-hand side yields:

$$\begin{aligned}\mathrm{dist}(\mathbf{G}_\theta^{\mathrm{next,AP}}, \mathbf{G}_\star) &\le \big\|\mathbf{I} - \eta \mathbf{F}_\theta{}^\top \mathbf{F}_\theta\big\|_{\mathrm{op}}\,\big\|\mathbf{G}_\theta \mathcal{P}_\star^\perp\big\|_{\mathrm{op}} \\ &= (1 - \eta\,\lambda_{\min}(\mathbf{F}_\theta{}^\top \mathbf{F}_\theta))\mathrm{dist}(\mathbf{G}_\theta, \mathbf{G}_\star),\end{aligned}$$

completing the one-step contraction bound for a step of AP.

To show that conversely $\mathbf{G}_\theta^{\mathrm{next,GD}}$ may in general be non-convergent in subspace distance, we construct a numerical example to illustrate this. Take $d = 1$, $n = 2$, and let

$$\mathbf{G}_\star = [1 \quad 0], \qquad \mathcal{P}_\star^\perp = \begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix}.$$

Choosing $\mathbf{G}_\theta = [\cos(\pi/3) \quad \sin(\pi/3)] = [1/2 \quad \sqrt{3}/2]$, we have

$$\mathrm{dist}(\mathbf{G}_\theta, \mathbf{G}_\star) = \|\mathbf{G}_\theta \mathcal{P}_\star^\perp\|_2 = \big\|[0 \quad \sqrt{3}/2]\big\|_2 = \frac{\sqrt{3}}{2}.$$

Now setting

$$\mathbf{\Sigma_s} = \begin{bmatrix}5 & 2\\ 2 & 1\end{bmatrix} \succ \mathbf{0}, \quad \mathbf{F}_\theta = 1, \quad \mathbf{F}_\star = 1, \quad \eta = 1,$$

we have

$$\mathbf{G}_\theta^{\mathrm{next,GD}}\mathcal{P}_\star^\perp = (\mathbf{G}_\theta - \mathbf{G}_\theta \mathbf{\Sigma_s})\mathcal{P}_\star^\perp + \mathbf{G}_\star \mathbf{\Sigma_s}\mathcal{P}_\star^\perp.$$

Computing the three terms:

$$\mathbf{G}_\theta \mathcal{P}_\star^\perp = \begin{bmatrix} 0 & \sqrt{3}/2 \end{bmatrix},$$
$$\mathbf{G}_\theta \Sigma_s \mathcal{P}_\star^\perp = \begin{bmatrix} 0 & 1+\sqrt{3}/2 \end{bmatrix}, \qquad \mathbf{G}_\star \Sigma_s \mathcal{P}_\star^\perp = \begin{bmatrix} 0 & 2 \end{bmatrix},$$

this yields

$$\begin{aligned} \mathbf{G}_\theta^{\text{next,GD}} \mathcal{P}_\star^\perp &= \begin{bmatrix} 0 & \sqrt{3}/2 \end{bmatrix} - \begin{bmatrix} 0 & 1+\sqrt{3}/2 \end{bmatrix} + \begin{bmatrix} 0 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \end{bmatrix}. \end{aligned}$$

Hence

$$\operatorname{dist}(\mathbf{G}_\theta^{\text{next,GD}}, \mathbf{G}_\star) = 1 > \frac{\sqrt{3}}{2} = \operatorname{dist}(\mathbf{G}_\theta, \mathbf{G}_\star),$$

completing the construction and the proof.

B.5 Dynamical Systems Theory

Proof of Proposition 3.1. We first establish the linear systems preliminaries in Proposition 3.1

Proposition 3.1. *Define the state-covariance matrix* $\mathbf{\Gamma}_t(\mathbf{K}) = \sum_{s=0}^{t-1} (\mathbf{A}+\mathbf{B}\mathbf{K})^s \mathbf{\Sigma}_\mathbf{w} (\mathbf{A}+\mathbf{B}\mathbf{K})^{s\top}$, *and* $\bar{\mathbf{\Gamma}}_t(\mathbf{K}) = \frac{1}{t}\sum_{s=1}^{t} \mathbf{\Gamma}_s(\mathbf{K})$. *Then,* $\mathbb{P}_{\mathbf{s}_t}^{\pi_{\text{demo}}} = \mathrm{N}(0, \mathbf{\Gamma}_t(\mathbf{K}_\star))$, $\mathbb{P}_{\hat{\mathbf{s}}_t}^{\pi_{\boldsymbol{\theta}}} = \mathrm{N}(0, \mathbf{\Gamma}_t(\mathbf{K}_\theta))$, *and* $\mathcal{L}_{\text{val}}(\pi_{\boldsymbol{\theta}}) = \left\|(\mathbf{K}_\star - \mathbf{K}_\theta)\bar{\mathbf{\Gamma}}_T(\mathbf{K}_\star)^{1/2}\right\|_{\mathrm{F}}^2$, $\mathcal{L}_{\text{ideal}}(\pi_{\boldsymbol{\theta}}) = -\mathcal{R}_{\text{test}}(\pi_{\boldsymbol{\theta}}) = \left\|(\mathbf{K}_\star - \mathbf{K}_\theta)\bar{\mathbf{\Gamma}}_T(\mathbf{K}_\theta)^{1/2}\right\|_{\mathrm{F}}^2$. *If* D *denotes the Wasserstein-2 distance, we have* $\mathrm{D}_t(\pi_{\boldsymbol{\theta}}) = \|\Gamma_t(\mathbf{K}_\star)^{1/2} - \Gamma_t(\mathbf{K}_\theta)^{1/2}\|_{\mathrm{F}}^2$.

Proof. Recall the state transition $\mathbf{s}_{t+1} = \mathbf{A}\mathbf{s}_t + \mathbf{B}\mathbf{a}_t + \mathbf{w}_t$, $\mathbf{w}_t \sim \mathrm{N}(\mathbf{0}, \mathbf{\Sigma}_\mathbf{w})$. Under $\mathbf{a}_t = \mathbf{K}\mathbf{s}_t$, we have for given $\mathbf{s}_t$:

$$\mathbf{s}_{t+1} \sim \mathrm{N}((\mathbf{A}+\mathbf{B}\mathbf{K})\mathbf{s}_t, \Sigma_\mathbf{W}).$$

Applying this recursively to $\mathbf{s}_0 = \mathbf{0}$ yields:

$$\mathbf{s}_t \sim \mathrm{N}\left(\mathbf{0}, \sum_{s=0}^{t-1} (\mathbf{A}+\mathbf{B}\mathbf{K})^s \mathbf{\Sigma}_\mathbf{w} (\mathbf{A}+\mathbf{B}\mathbf{K})^{s\top}\right) =: \mathrm{N}(\mathbf{0}, \Gamma_t(\mathbf{K})).$$

Plugging in $\mathbf{K} = \mathbf{K}_\star, \mathbf{K}_\theta$ completes the proof of the distribution of the state marginals: $\mathbb{P}_{\mathbf{s}_t}^{\pi_{\text{demo}}} = \mathrm{N}(0, \mathbf{\Gamma}_t(\mathbf{K}_\star))$, $\mathbb{P}_{\hat{\mathbf{s}}_t}^{\pi_{\boldsymbol{\theta}}} = \mathrm{N}(0, \mathbf{\Gamma}_t(\mathbf{K}_\theta))$. As for the losses, we have:

$$\begin{aligned} \mathcal{L}_{\text{val}}(\pi_{\boldsymbol{\theta}}) &= \mathbb{E}^{\pi_{\text{demo}}} \|\mathbf{K}_\theta \mathbf{s} - \mathbf{a}\|_2^2 \\ &= \mathbb{E}^{\pi_{\text{demo}}} \frac{1}{T}\sum_{t=1}^{T} \|(\mathbf{K}_\theta - \mathbf{K}_\star)\mathbf{s}_t\|_2^2 \\ &= \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}^{\pi_{\text{demo}}} \operatorname{tr}\left((\mathbf{K}_\theta - \mathbf{K}_\star)\mathbf{s}_t\mathbf{s}_t^\top(\mathbf{K}_\theta - \mathbf{K}_\star)^\top\right) \\ &= \operatorname{tr}\left((\mathbf{K}_\theta - \mathbf{K}_\star)\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}^{\pi_{\text{demo}}}[\mathbf{s}_t\mathbf{s}_t^\top](\mathbf{K}_\theta - \mathbf{K}_\star)^\top\right) \\ &= \operatorname{tr}\left((\mathbf{K}_\theta - \mathbf{K}_\star)\bar{\mathbf{\Gamma}}_t(\mathbf{K}_\star)(\mathbf{K}_\theta - \mathbf{K}_\star)^\top\right) \\ &= \|(\mathbf{K}_\theta - \mathbf{K}_\star)\bar{\mathbf{\Gamma}}_t(\mathbf{K}_\star)^{1/2}\|_F^2, \\ \mathcal{L}_{\text{ideal}}(\pi_{\boldsymbol{\theta}}) &= \mathbb{E}^{\pi_{\boldsymbol{\theta}}} \|\mathbf{K}_\theta \hat{\mathbf{s}} - \hat{\mathbf{a}}\|_2^2 \\ &= \operatorname{tr}\left((\mathbf{K}_\theta - \mathbf{K}_\star)\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}^{\pi_{\boldsymbol{\theta}}}[\hat{\mathbf{s}}_t\hat{\mathbf{s}}_t^\top](\mathbf{K}_\theta - \mathbf{K}_\star)^\top\right) \\ &= \|(\mathbf{K}_\theta - \mathbf{K}_\star)\bar{\mathbf{\Gamma}}_t(\mathbf{K}_\theta)^{1/2}\|_F^2. \end{aligned}$$

Finally, when the divergence D is the Wasserstein-2 distance, we may use the formula for Wasserstein-2 distance between two multivariate Gaussians $\mathbb{P} = \mathrm{N}(\mathbf{0}, \mathbf{\Sigma}_1)$, $\mathbb{Q} = \mathrm{N}(\mathbf{0}, \mathbf{\Sigma}_2)$:

$$\mathcal{W}_2^2(\mathbb{P}, \mathbb{Q}) = \operatorname{tr}(\mathbf{\Sigma}_1 + \mathbf{\Sigma}_2 - 2(\mathbf{\Sigma}_1^{1/2}\mathbf{\Sigma}_2\mathbf{\Sigma}_1^{1/2})^{1/2}).$$

Plugging in $\mathbf{\Sigma}_1 = \bar{\mathbf{\Gamma}}_t(\mathbf{K}_\star)$, $\mathbf{\Sigma}_2 = \bar{\mathbf{\Gamma}}_t(\mathbf{K}_\theta)$ and simplifying yields the result and completes the proof.

□

B.5.1 MISMATCH BETWEEN $\mathcal{L}_{\text{ideal}}$ AND $\mathcal{L}_{\text{val}}$: A CONSTRUCTION

We want to demonstrate that due to feedback between a policy and the dynamics, two policies that may satisfy $\mathcal{L}_{\text{val}}(\pi_1) \ll \mathcal{L}_{\text{val}}(\pi_2)$ may in fact satisfy $\mathcal{L}_{\text{ideal}}(\pi_1) \gg \mathcal{L}_{\text{ideal}}(\pi_2)$. Consider the dynamical system

$$\mathbf{A} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \mathbf{I}, \quad \mathbf{\Sigma_w} = \mathbf{I} \tag{B.4}$$

Next, we set

$$\mathbf{K}_\star = \begin{bmatrix} 0 & 0 \\ 0 & -\alpha \end{bmatrix}, \quad \mathbf{F}_\star = \mathbf{e}_2, \quad \mathbf{G}_\star = -\alpha \mathbf{e}_2^\top \tag{B.5}$$

Finally, let us consider two learned parameters

$$\mathbf{K}_\theta = \mathbf{F}_\theta \mathbf{G}_\theta, \quad \mathbf{F}_\theta = \mu \mathbf{e}_2, \quad \mathbf{G}_\theta = -\alpha \mu^{-1} \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}^\top \tag{B.6}$$

$$\mathbf{K}_\psi = \mathbf{F}_\psi \mathbf{G}_\psi, \quad \mathbf{F}_\psi = \begin{bmatrix} \alpha \\ 1 \end{bmatrix}, \quad \mathbf{G}_\psi = \mathbf{G}_\star \tag{B.7}$$

Then, we have that

$$\mathbf{K}_\theta = \mathbf{K}_\star - \alpha \begin{bmatrix} 0 & 0 \\ -\cos\theta & 1 - \sin\theta \end{bmatrix} \tag{B.8}$$

$$\mathbf{K}_\psi = \mathbf{K}_\star - \epsilon \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{B.9}$$

And the following computation will be useful:

Lemma B.2. *It holds that*

$$(\mathbf{K}_\theta - \mathbf{K}_\star)^\top (\mathbf{K}_\theta - \mathbf{K}_\star) = \alpha^2 \begin{bmatrix} \cos^2\theta & -\cos\theta(1-\sin\theta) \\ -\cos\theta(1-\sin\theta) & (1-\sin\theta)^2 \end{bmatrix} \tag{B.10}$$

$$(\mathbf{K}_\psi - \mathbf{K}_\star)^\top (\mathbf{K}_\psi - \mathbf{K}_\star) = \begin{bmatrix} 0 & 0 \\ 0 & \epsilon^2 \end{bmatrix} \tag{B.11}$$

For simplicity, we work now in the limit where $T \to \infty$. The following is standard:

Lemma B.3. *Let* $\mathbf{K}$ *be such that* $\rho(\mathbf{A} + \mathbf{BK}) < 1$, *where* $\rho(\cdot)$ *denotes the spectral radius. Then,* $\mathbf{\Gamma}_\infty(\mathbf{K}) := \lim_{T\to\infty} \mathbf{\Gamma}_T(\mathbf{K})$ *exists and is finite, is equal to* $\lim_{T\to\infty} \bar{\mathbf{\Gamma}}_T(\mathbf{K})$, *and is given by the unique solution to the following Riccati equation:*

$$\mathbf{\Gamma}_\infty(\mathbf{K}) = \mathbf{\Sigma_w} + (\mathbf{A} + \mathbf{BK}) \mathbf{\Gamma}_\infty(\mathbf{K}) (\mathbf{A} + \mathbf{BK})^\top. \tag{B.12}$$

We now compute the stationary covariances under $\mathbf{K}_\star, \mathbf{K}_\theta, \mathbf{K}_\psi$.

Lemma B.4. $\mathbf{\Gamma}_\infty(\mathbf{K}_\star)$ *exists, and is given by*

$$\mathbf{\Gamma}_\infty(\mathbf{K}_\star) = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{1-(1-\alpha)^2} \end{bmatrix} \tag{B.13}$$

Proof. $\mathbf{A} + \mathbf{BK}_\star$ has eigenvalues $0.5, 1-\alpha$, so Lemma B.3 holds. In this case, the limiting Γ satisfies

$$\mathbf{\Gamma} = \begin{bmatrix} 1 & 0 \\ 0 & 1-\alpha \end{bmatrix} \mathbf{\Gamma} \begin{bmatrix} 0 & 0 \\ 0 & 1-\alpha \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{B.14}$$

decoupling across coordinates. So we select $\mathbf{\Gamma}$ to be diagonal with coordinates γ_1, γ_2 on the diagonals. These must satisfy γ_1, and $\gamma_2(1-\alpha)^2 + 1 = \gamma_2$, so that $\gamma = 1/(1-(1-\alpha)^2)$ □

Lemma B.5. *For $\theta \in (0, \pi/2)$, $\mathbf{\Gamma}_\infty(\mathbf{K}_\theta)$ exists,*

$$\mathbf{\Gamma}_\infty(\mathbf{K}_\theta) = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1+\alpha^2\cos^2\theta}{\alpha\sin\theta(2-\alpha\sin\theta)} \end{bmatrix} \tag{B.15}$$

Proof of Lemma B.5. The spectral radius condition can be checked for $\theta \in (0, \pi/2)$ by upper triangularity. Now set

$$\mathbf{\Gamma} = \begin{bmatrix} 0 & 0 \\ -\alpha\cos\theta & 1-\alpha\sin\theta \end{bmatrix} \mathbf{\Gamma} \begin{bmatrix} 0 & 0 \\ -\alpha\cos\theta & 1-\alpha\sin\theta \end{bmatrix}^\top + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{B.16}$$

We now solve the above:

$$\mathbf{X} := \begin{bmatrix} 0 & 0 \\ -\alpha\cos\theta & 1-\alpha\sin\theta \end{bmatrix}, \qquad \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Now write

$$\mathbf{\Gamma} = \begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{12} & \gamma_{22} \end{bmatrix}.$$

The Lyapunov equation to solve is $\mathbf{\Gamma} = \mathbf{X}\mathbf{\Gamma}\mathbf{X}^\top + Q$. Computing

$$\mathbf{X}\mathbf{\Gamma}\mathbf{X}^\top = \begin{bmatrix} 0 & 0 \\ 0 & \alpha^2\cos^2\theta\,\gamma_{11} - 2\alpha\cos\theta\,(1-\alpha\sin\theta)\,\gamma_{12} + (1-\alpha\sin\theta)^2\,\gamma_{22} \end{bmatrix}.$$

Matching entries gives

$$\begin{aligned} &\gamma_{11} = 1, \\ &\gamma_{12} = 0, \\ &\gamma_{22} = \alpha^2\cos^2\theta\,\gamma_{11} - 2\alpha\cos\theta\,(1-\alpha\sin\theta)\,\gamma_{12} + (1-\alpha\sin\theta)^2\,\gamma_{22} + 1. \end{aligned}$$

Using $\gamma_{11} = 1$ and $\gamma_{12} = 0$, the final equation becomes

$$\gamma_{22} = \alpha^2\cos^2\theta + (1-\alpha\sin\theta)^2\,\gamma_{22} + 1.$$

Therefore, $\left[1-(1-\alpha\sin\theta)^2\right]\gamma_{22} = 1 + \alpha^2\cos^2\theta$, and since $1-(1-\alpha\sin\theta)^2 = \alpha\sin\theta\,(2-\alpha\sin\theta)$, we obtain

$$\gamma_{22} = \frac{1+\alpha^2\cos^2\theta}{\alpha\sin\theta\,(2-\alpha\sin\theta)}.$$

Hence

$$\mathbf{\Gamma} = \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{1+\alpha^2\cos^2\theta}{\alpha\sin\theta\,(2-\alpha\sin\theta)} \end{bmatrix}.$$

□

Lemma B.6. *$\mathbf{\Gamma}_\infty(\mathbf{K}_\psi)$ exists, and satisfies*

$$\mathbf{\Gamma}_\infty(\mathbf{K}_\psi) = \begin{bmatrix} 1+\frac{\epsilon^2}{\alpha(2-\alpha)} & -\frac{\epsilon(1-\alpha)}{\alpha(2-\alpha)} \\ -\frac{\epsilon(1-\alpha)}{\alpha(2-\alpha)} & \frac{1}{\alpha(2-\alpha)} \end{bmatrix}.$$

Proof of Lemma B.6. Again, we can check the stability conditions from upper tri-angularity. We have

$$\mathbf{\Gamma} = \underbrace{\begin{bmatrix} 0 & -\epsilon \\ 0 & 1-\alpha \end{bmatrix}}_{:=\mathbf{X}} \mathbf{\Gamma} \begin{bmatrix} 0 & -\epsilon \\ 0 & 1-\alpha \end{bmatrix}^\top + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{B.17}$$

$$\mathbf{X} := \begin{bmatrix} 0 & -\epsilon \\ 0 & 1-\alpha \end{bmatrix}, \qquad \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Now writing $\mathbf{\Gamma} = \begin{bmatrix} \gamma_{11} & \gamma_{12} \\ \gamma_{12} & \gamma_{22} \end{bmatrix}$, the Lyapunov equation to solve is $\mathbf{\Gamma} = \mathbf{X}\mathbf{\Gamma}\mathbf{X}^\top + \mathbf{Q}$. First, we compute

$$\mathbf{X}\mathbf{\Gamma}\mathbf{X}^\top = \begin{bmatrix} \epsilon^2\gamma_{22} & -\epsilon(1-\alpha)\gamma_{22} \\ -\epsilon(1-\alpha)\gamma_{22} & (1-\alpha)^2\gamma_{22} \end{bmatrix}.$$

Matching entries gives

$$\begin{aligned} \gamma_{11} &= \epsilon^2\gamma_{22} + 1, \\ \gamma_{12} &= -\epsilon(1-\alpha)\gamma_{22}, \\ \gamma_{22} &= (1-\alpha)^2\gamma_{22} + 1. \end{aligned}$$

We solve from bottom to top. The third equation gives $\left(1-(1-\alpha)^2\right)\gamma_{22} = 1$. Since $1-(1-\alpha)^2 = \alpha(2-\alpha)$, we obtain $\gamma_{22} = \frac{1}{\alpha(2-\alpha)}$. Therefore,

$$\gamma_{12} = -\epsilon(1-\alpha)\gamma_{22} = -\frac{\epsilon(1-\alpha)}{\alpha(2-\alpha)}\gamma_{11} \qquad = 1 + \epsilon^2\gamma_{22} = 1 + \frac{\epsilon^2}{\alpha(2-\alpha)}.$$

Hence

$$\mathbf{\Gamma} = \begin{bmatrix} 1 + \dfrac{\epsilon^2}{\alpha(2-\alpha)} & -\dfrac{\epsilon(1-\alpha)}{\alpha(2-\alpha)} \\ -\dfrac{\epsilon(1-\alpha)}{\alpha(2-\alpha)} & \dfrac{1}{\alpha(2-\alpha)} \end{bmatrix}.$$

□

From here, we can compute the various error terms

Lemma B.7. *Let $\theta \in (0, \pi/4)$. Then, taking the $T \to \infty$ limit,*

$$\mathcal{L}_{\text{val}}(\mathbf{K}_\theta) \sim \alpha, \quad \mathcal{L}_{\text{ideal}}(\mathbf{K}_\theta) \sim \frac{\alpha}{\sin\theta}, \tag{B.18}$$

where as

$$\mathcal{L}_{\text{val}}(\mathbf{K}_\psi), \mathcal{L}_{\text{ideal}}(\mathbf{K}_\psi) \sim \frac{\epsilon^2}{\alpha} \tag{B.19}$$

Proof. From Proposition 3.1, the fact that $\|\Delta\mathbf{\Gamma}^{1/2}\|_{\text{F}}^2 = \text{tr}(\Delta^\top\Delta\mathbf{\Gamma})$, and that $\Delta = \mathbf{K}_\star - \mathbf{K}_\theta$, we have that $\text{tr}(\Delta^\top\Delta\mathbf{\Gamma}) = \alpha^2\mathbf{\Gamma}_{22}(1-\sin\theta)^2$ by Lemma B.2, we obtain

$$\mathcal{L}_{\text{val}}(\mathbf{K}_\theta) = \alpha^2\cos^2\theta + \alpha^2(1-\sin\theta)^2 \cdot \frac{1}{1-(1-\alpha)^2} \sim \alpha \tag{B.20}$$

for α small and $\theta \in (0, \pi/4)$. Moreover, from these same argument

$$\mathcal{L}_{\text{ideal}}(\mathbf{K}_\theta) = \alpha^2\cos^2\theta + \alpha^2(1-\sin\theta)^2 \cdot \frac{1+\alpha^2\cos^2\theta}{\alpha\sin\theta(2-\alpha\sin\theta)} \sim \alpha^2 + \frac{\alpha}{\sin\theta} \sim \frac{\alpha}{\sin\theta}.$$

We may compute the quantities on $\mathbf{K}_\psi$ similarly. □

By modulating ϵ, α appropriately, the following proposition follows immediately:

Proposition B.8. *For $\epsilon \ll \alpha$ and $\sin\theta \ll \epsilon^2/\alpha^2$, we have we have that*

$$\mathcal{L}_{\text{val}}(\mathbf{K}_\theta) \ll \mathcal{L}_{\text{val}}(\mathbf{K}_\psi) \sim \mathcal{L}_{\text{ideal}}(\mathbf{K}_\psi) \ll \mathcal{L}_{\text{ideal}}(\mathbf{K}_\theta). \tag{B.21}$$

Continuing with these quantities, we may go on to prove Proposition 3.2.

B.5.2 Proof of Proposition 3.2

We recall Proposition 3.2:

Proposition 3.2 (Poor Feature Learning causes TTF Shift, Informal). *There exists $(\mathbf{F}_1, \mathbf{G}_1)$, $(\mathbf{F}_2, \mathbf{G}_2)$, $(\mathbf{F}_\star, \mathbf{G}_\star)$ and linear dynamics $\mathbf{A}, \mathbf{B}, \mathbf{\Sigma}_{\mathbf{w}}$ such that $\mathcal{L}_{\text{val}}(\mathbf{F}_1\mathbf{G}_1) \ll \mathcal{L}_{\text{val}}(\mathbf{F}_2\mathbf{G}_2)$, but $\text{dist}(\mathbf{G}_1, \mathbf{G}_\star) > \text{dist}(\mathbf{G}_2, \mathbf{G}_\star)$, and for any possible "last-layer" $\mathbf{F}_\theta$, we have $\mathcal{L}_{\text{ideal}}(\mathbf{F}_2\mathbf{G}_2) \ll \mathcal{L}_{\text{ideal}}(\mathbf{F}_\theta\mathbf{G}_1)$.*

Notably, compared to the constructions above, the difference is that Proposition 3.2 introduces the additional complication of allowing the flexibility of re-fitting the last-layer $\mathbf{F}_\theta$. Inheriting $\mathbf{K}_\star, \mathbf{K}_\theta, \mathbf{K}_\psi$ and relevant quantities therein from earlier, the key result to establish is:

Lemma B.9. *For $\theta \in (0, \pi/8)$, we have $\inf_{\mathbf{F}} \mathcal{L}_{\text{ideal}}(\mathbf{F}\mathbf{G}_\theta) \geq \frac{\alpha^2}{4\sin\theta}$.*

By rescaling $\mathbf{G}_\theta, \mathbf{F}_\theta$ by μ^{-1} and μ, and using Proposition 3.1 as in the proof of Lemma B.7, it suffices to prove the following.

Lemma B.10. *Let $\theta \leq \pi/8$. Define the vector*

$$\mathbf{w} = -\alpha \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}^\top, \quad \mathbf{A} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{B.22}$$

For any $\mathbf{v}$, define two terms: $\mathbf{X}_\mathbf{v} = \mathbf{A} - \mathbf{v}\mathbf{w}^\top$, $\Delta_\mathbf{v} = \alpha\mathbf{A} - \mathbf{v}\mathbf{w}^\top$. When $\mathbf{v}$ is such that a solution $\mathbf{\Gamma}_\mathbf{v}$ to the Riccati equation

$$\mathbf{\Gamma}_\mathbf{v} = \mathbf{X}_\mathbf{v}\mathbf{\Gamma}_\mathbf{v}\mathbf{X}_\mathbf{v}^\top + \mathbf{Q}. \tag{B.23}$$

exists, where $\mathbf{\Gamma}_\mathbf{v}$ is defined as above. Then, we have

$$\inf_{\mathbf{v}\in\mathbb{R}^2 : \mathbf{\Gamma}_\mathbf{v} \text{ exists}} \text{tr}\left(\Delta_\mathbf{v}^\top \Delta_\mathbf{v} \cdot \mathbf{\Gamma}_\mathbf{v}\right) \geq \frac{\alpha^2}{4\sin\theta}. \tag{B.24}$$

Proof.

$$\mathbf{w} = -\alpha \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \qquad \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

For any $\mathbf{v} \in \mathbb{R}^2$, define $\mathbf{X}_\mathbf{v} = \mathbf{A} - \mathbf{v}\mathbf{w}^\top$ and $\Delta_\mathbf{v} = \alpha\mathbf{A} - \mathbf{v}\mathbf{w}^\top$. Since $-\mathbf{v}\mathbf{w}^\top = \alpha\mathbf{v}\begin{bmatrix}\cos\theta & \sin\theta\end{bmatrix}$, we have

$$\mathbf{X}_\mathbf{v} = \mathbf{A} + \alpha\mathbf{v}\begin{bmatrix}\cos\theta & \sin\theta\end{bmatrix}, \qquad \Delta_\mathbf{v} = \alpha\mathbf{A} + \alpha\mathbf{v}\begin{bmatrix}\cos\theta & \sin\theta\end{bmatrix}.$$

Let $c := \cos\theta$, $s := \sin\theta$, and $\tau := \tan\theta = \frac{s}{c}$. Writing $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, define $p := \alpha v_1$ and $q := \alpha v_2$. Then

$$\mathbf{X}_\mathbf{v} = \begin{bmatrix} pc & ps \\ qc & 1+qs \end{bmatrix}, \qquad \Delta_\mathbf{v} = \begin{bmatrix} pc & ps \\ qc & \alpha+qs \end{bmatrix}.$$

Now reparameterize by $r := pc = \alpha v_1 \cos\theta$ and $h := -\alpha v_2 \sin\theta\theta$. Since $s, c > 0$, this is an equivalent parameterization. In these variables,

$$\mathbf{X}_\mathbf{v} = \begin{bmatrix} r & r\tau \\ -\frac{h}{\tau} & 1-h \end{bmatrix}, \qquad \Delta_\mathbf{v} = \begin{bmatrix} r & r\tau \\ -\frac{h}{\tau} & \alpha-h \end{bmatrix}.$$

The characteristic polynomial of $\mathbf{X_v}$ is $\chi(z) = z^2 - (1 + r - h)z + r$; therefore, by the Schur stability conditions, the Lyapunov/Riccati solution $\mathbf{\Gamma_v} = \mathbf{X_v}\mathbf{\Gamma_v}\mathbf{X_v}^\top + \mathbf{Q}$ exists if and only if $r < 1$, $h > 0$, and $2 + 2r - h > 0$. Equivalently, $-1 < r < 1$ and $0 < h < 2(1 + r)$. Define the objective $J(r, h) := \operatorname{tr}\left(\Delta_\mathbf{v}^\top \Delta_\mathbf{v} \mathbf{\Gamma_v}\right)$. Solving the Lyapunov equation and substituting gives

$$J(r,h) = \frac{N(r,h,\tau,\alpha)}{h\tau^2(1-r)(2+2r-h)},$$

where

$$\begin{aligned} N(r,h,\tau,\alpha) = {} & \alpha^2(1+r)h^2 + \alpha^2\tau^2(1-r)^2(1+r) + 2(1-\alpha)h^3 \\ & + \tau^2\left[2\alpha^2 hr - 2\alpha h^2 r + 2\alpha h r^2 - 2\alpha h + h^2(1+r) + 2hr^2\right] \\ & + \tau^4 r^2(1+r). \end{aligned}$$

For fixed $0 < \alpha \leq 1$ and θ sufficiently small, the numerator satisfies the lower bound

$$N(r,h,\tau,\alpha) \geq \frac{\alpha^2}{2}(1+r)\left[h^2 + \tau^2(1-r)^2\right].$$

Also, since $0 < h < 2(1 + r)$, we have $2 + 2r - h \leq 2(1 + r)$. Therefore,

$$\begin{aligned} J(r,h) &\geq \frac{\frac{\alpha^2}{2}(1+r)\left[h^2 + \tau^2(1-r)^2\right]}{h\tau^2(1-r)\,2(1+r)} \\ &= \frac{\alpha^2}{4}\frac{h^2 + \tau^2(1-r)^2}{h\tau^2(1-r)}. \end{aligned}$$

By AM-GM, $h^2 + \tau^2(1 - r)^2 \geq 2h\tau(1 - r)$. Hence

$$\begin{aligned} J(r,h) &\geq \frac{\alpha^2}{4} \cdot \frac{2h\tau(1-r)}{h\tau^2(1-r)} \\ &= \frac{\alpha^2}{2\tau}. \end{aligned}$$

Thus,

$$\inf_{\mathbf{v}:\,\mathbf{\Gamma_v}\text{ exists}} \operatorname{tr}\left(\Delta_\mathbf{v}^\top \Delta_\mathbf{v} \mathbf{\Gamma_v}\right) \geq \frac{\alpha^2}{2\tan\theta}.$$

Since $\tan\theta \leq 2\sin\theta$ for θ sufficiently small, this further implies

$$\inf_{\mathbf{v}:\,\mathbf{\Gamma_v}\text{ exists}} \operatorname{tr}\left(\Delta_\mathbf{v}^\top \Delta_\mathbf{v} \mathbf{\Gamma_v}\right) \geq \frac{\alpha^2}{4\sin\theta}.$$

□

Thus, chaining Proposition B.8 and Lemma B.9 together yields the following quantitative form of Proposition 3.2.

Proposition B.11. *For* $\sin\theta \ll \frac{\epsilon^2}{\alpha^3}$ *and* $\epsilon \ll \alpha$, *we have*

$$\mathcal{L}_{\text{val}}(\mathbf{K}_\theta) \ll \mathcal{L}_{\text{val}}(\mathbf{K}_\psi) \sim \mathcal{L}_{\text{ideal}}(\mathbf{K}_\psi) \ll \inf_{\mathbf{F}} \mathcal{L}_{\text{ideal}}(\mathbf{F}\mathbf{G}_\theta) \leq \mathcal{L}_{\text{ideal}}(\mathbf{K}_\theta). \tag{B.25}$$

B.6 Computational Overhead

`DoPr` incurs a computational and memory overhead via the activation covariance. For a linear layer with input dimension d_{in}, output dimension d_{out}, and batch size N, the per-step overhead consists of

accumulating the input covariance, factoring the damped preconditioner, and applying the inverse preconditioner to the gradient:

$$\mathbf{S} = \mathbf{X}^\top \mathbf{X}, \qquad \mathbf{X} \in \mathbb{R}^{B \times d_{\text{in}}},$$

which costs approximately Bd_{in}^2 FLOPs (due to symmetry of $\mathbf{S}$. Computing the Cholesky factorization of the damped matrix $\mathbf{A} = \mathbf{S} + \lambda I \in \mathbb{R}^{d_{\text{in}} \times d_{\text{in}}}$ costs approximately $\frac{1}{3}d_{\text{in}}^3$ FLOPs. Finally, applying the inverse preconditioner to the weight gradient

$$\mathbf{G}(\mathbf{S} + \lambda \mathbf{I})^{-1}, \qquad \mathbf{G} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}},$$

via two triangular solves costs $2d_{\text{in}}^2 d_{\text{out}}$ FLOPs. Thus, the total per-layer per-step idealized FLOP overhead is

$$\boxed{Bd_{\text{in}}^2 + \frac{1}{3}d_{\text{in}}^3 + 2d_{\text{in}}^2 d_{\text{out}}}.$$

In line with empirical know-how (see e.g., Jordan et al. (2024)), in the large-batch regime, the cost of forming the activation covariance $\mathbf{S} = \mathbf{X}^\top \mathbf{X}$ (the first term) dominates. Thus, for layers where $d_{\text{in}} \approx d_{\text{out}}$, the overhead is approximately equal to an additional forward pass. Indeed, throughout our experiments beyond small models (wherein `DoPr`'s overhead like other matrix-shaped preconditioners is compute-bound rather than memory-bound), we found the overhead upper bounded by $\approx 1.25x$ in wall-clock time, commensurate with the estimate of a backward pass equal to ~ 3 forward passes, such that `AP`'s overhead coarsely comes out to $5/4 = 1.25\times$ regular backprop. Furthermore, this overhead is exact only when a feedforward layer has unique activations. This overhead becomes amortized when dealing with layer-types that share an activation stream, such as attention QKV parameterizations. This is the same observation made in Du & Su (2026) and Hong & Qu (2026), where the effective per-step overhead can be as low as 10%, which can only be further reduced as architectures which increase parameter-to-activation (such as Multi-Head Attention or Grouped-Query Attention) are popularized.

`DoPr` can also introduce additional memory overhead from storing the matrix-valued preconditioner statistics. With an exponential moving average (EMA), this requires storing one $d_{\text{in}} \times d_{\text{in}}$ symmetric matrix per preconditioned layer, i.e., $\sim d_{\text{in}}^2/2$ additional parameters of optimizer state per layer. However, we note that across the majority of our experiments, we found the batch-wise preconditioner (i.e. requiring no additional persistent memory) to be sufficient if not optimal.

Our current implementation of `DoPr` supports Distributed Data Parallelism, and not model parameter sharding such as FSDP. However, we do not anticipate any fundamental barriers for such schemes beyond existing related matrix-preconditioning methods such as `Shampoo`, which have distributed and sharded implementations (Shi et al., 2023; Ahn et al., 2025b). Finally, `DoPr` like other matrix preconditioners that involve `linsolve`'s can be sensitive to the choice of matrix-inversion solver and damping parameter. In practice, we found PyTorch's Cholesky solve together with trace-scaled damping to provide a sufficiently stable and effective default.

`DoPr` may improve downstream performance without improving training or validation loss, and it may not improve training-loss convergence relative to the base `GP` optimizer. Moreover, exact `AP` for convolutional or other weight-sharing layers can incur substantial memory overhead due to unfolding. While we introduce approximation schemes to account for the redundancy therein, developing hardware-accelerated approximations and exploring model–optimizer codesign remain important directions for future work.

C Experiment Details and Additional Results

Hardware

- All imitation learning experiments were performed on 4× NVIDIA RTX A5000 GPUs.
- All language modeling experiments were performed on 8× NVIDIA RTX PRO 6000 Blackwell GPUs.
- All image generation experiments were performed on 8× NVIDIA A100 GPUs.

C.1 General Operating Guideline

In practice, we summarize a small set of operating guidelines that we have found to be broadly useful across settings.

- **EMA smoothing for very small-batch, continuous-valued inputs.** For continuous-valued inputs (in contrast to discrete-token embeddings in LLMs), we suggest applying to the `AP` an exponential moving average (EMA) to both the activation covariance and gradient to reduce update variance when the batch size is significantly smaller than the model width. We note that appropriate considerations should be made if the `GP` of choice also contains EMAs, as this may doubly apply smoothing on the gradient.
- **Use no/low dropout.** Dropout (Srivastava et al., 2014) is designed to prevent spurious co-adaptation between features in the network. Firstly, due to masking different subsets of activations per update, large dropout can conflict with accumulation in the activation preconditioner buffers, accelerating staleness. Furthermore, `AP` can be viewed as similarly de-biasing the gradient from its activation (i.e. feature) correlations. Therefore, we recommend using no dropout for most settings, or reduced amounts in large-scale convolutional architectures.
- **Handling high-redundancy/weight-sharing layers.** `AP` is most directly derived for linear layers. For layers that have high-redundancy and weight-sharing properties, such as convolutional layers, the exact `AP` (resulting from Proposition 4.2) can incur needlessly large memory overhead from unfolding. Following prior literature in Kronecker-Factored preconditioners (Grosse & Martens, 2016; George et al., 2018), we may derive an efficient approximation of `AP` that empirically preserves the performance of the full `AP` as in Appendix B.3.2. We also note that one of the motivations of heavy weight-sharing is to make the *architecture* "well-conditioned" (Wilson et al., 2017; Keskar & Socher, 2017; Xiao et al., 2018), and thus may be partially redundant to our optimizer that is designed to handle heavy anisotropy. As we do not modify baseline model architectures in this work, we leave exploring model-optimizer co-design for future work.

Before proceeding, we remark that the experiments in our paper intentionally focus on `DoPr`'s effectiveness with settings that experience TTF, such as off-policy learning (SFT, imitation learning), in contrast to settings like RLVR where TTF is directly tackled by performing *on-policy* updates, i.e. the data distribution matches the rollout distribution, at the cost of blowing up inference costs due to online rollouts. That said, we do see that issues related to TTF occur in RLVR, e.g. trainer-inference mismatch, though in those specific cases the relevant interventions are more to do with update frequency and precision format. Our derivations motivate that `DoPr`'s core contribution is improvement in the feature learning and thus downstream performance, possibly independently of its loss convergence (i.e. the focus of pre-training literature). That said, we refer to Du & Su (2026); Hong & Qu (2026) for concurrent works that derive subcases of `DoPr` for speeding up LLM pre-training.

C.2 Feature Learning: Validation Loss versus Subspace Distance

This section provides details on the experiments shown in Figure 3.

Data generation. We follow the synthetic two-layer setup of Zhang et al. (2025a). We first generate a base matrix $\mathbf{F}_0 \in \mathbb{R}^{\mathsf{d_Y} \times k}$ with i.i.d. $\mathsf{N}(0,1)$ entries. For each task $\mathbf{s} \in \{\mathsf{train}, \mathsf{test}\}$, we sample a matrix $\mathbf{B_s} \in \mathbb{R}^{\mathsf{d_Y} \times \mathsf{d_Y}}$ and define the task-specific head

$$\mathbf{F}_\star^{\mathbf{s}} = \exp\big(0.005(\mathbf{B_s} - \mathbf{B_s}^\top)\big)\,\mathbf{F}_0,$$

which produces small random rotations of a shared base feature matrix. The shared representation $\mathbf{G}_\star \in \mathbb{R}^{k \times \mathsf{d_X}}$ is sampled uniformly from the set of row-orthonormal matrices. For each task $\mathbf{s}$, inputs are generated as

$$\mathbf{x}_i^{\mathbf{s}} \sim \mathbf{\Sigma}_{\mathbf{x},\mathbf{s}}^{1/2} \cdot \mathrm{Unif}(\{\pm 1\}^{\mathsf{d_X}}),$$

and outputs are generated according to

$$\mathbf{y}_i^{\mathbf{s}} = \mathbf{F}_\star^{\mathbf{s}} \mathbf{G}_\star \mathbf{x}_i^{\mathbf{s}} + \boldsymbol{\varepsilon}_i^{\mathbf{s}}, \quad \boldsymbol{\varepsilon}_i^{\mathbf{s}} \sim \mathsf{N}(0, \sigma_{\varepsilon,\mathbf{s}}^2 \mathbf{I}_{\mathsf{d_Y}}).$$

Anisotropy. We sample a random rotation $\mathbf{O} \in \mathbb{R}^{\mathsf{d_X} \times \mathsf{d_X}}$ and define

$$\mathbf{\Sigma}_{\mathbf{x},\mathbf{s}} = \mathbf{O}\mathbf{D}\mathbf{O}^\top, \quad \mathbf{D} = \mathbf{diag}(\texttt{logspace}(0, 5, \mathsf{d_X})),$$

yielding highly ill-conditioned covariances. This setting amplifies the feature-learning bias described in Proposition 4.1.

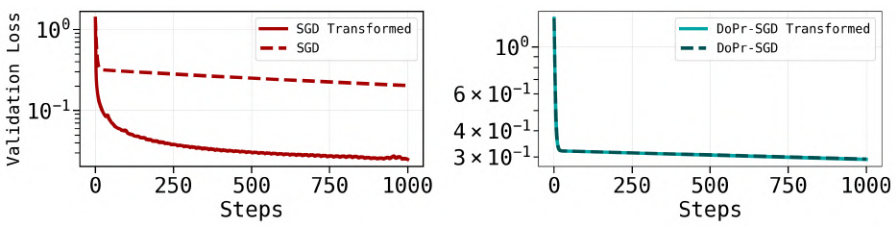


Figure 9: When an affine transform is applied to the input distribution, with the initial weights transformed accordingly (4.2). The SGD trajectories (**left**) diverge, while the DoPr-SGD trajectories (**right**) match exactly, demonstrating the invariance induced by DoPr under affine transforms (Proposition 4.2). See Appendix C.3 for experiment details.

Model and training. We train linear models of the form $\mathbf{F}_\theta \mathbf{G}_\theta$ on the training task train using squared loss, i.e.,

$$\mathcal{L}_{\text{train}}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}_{\text{train}}} \left[\|\mathbf{F}_\theta \mathbf{G}_\theta \mathbf{x} - \mathbf{y}\|^2 \right].$$

We compare SGD, Adam, and DoPr-SGD, using batch size $n = 1024$ and averaging results over 10 random seeds. All methods are initialized identically and use matched learning rates.

Metrics. We report (i) the in-distribution validation loss $\mathcal{L}_{\text{val}}$ on held-out samples from train, and (ii) the subspace distance $\text{dist}(\mathbf{G}_\theta, \mathbf{G}_\star)$, which measures alignment between the learned and ground-truth feature subspaces. The latter isolates representation learning independent of downstream fitting.

Results. As shown in Figure 3, validation loss suggests successful optimization across all methods. However, the subspace distance reveals a clear discrepancy: standard and gradient-preconditioned methods fail to recover the true feature subspace under high anisotropy, whereas AP significantly improves alignment. This demonstrates that low validation loss can mask poor feature learning, and supports the claim that AP mitigates TTF by promoting more uniform feature recovery.

C.3 Activation Invariance of DoPr

This section provides details on the experiments shown in Figure 9.

Models and datasets. We use 3 layer MLPs trained on 2 synthetically generated datasets. For the first dataset, the input data is generated with an anisotropic covariance $\mathbf{\Sigma} \in \mathbb{R}^{32\times32}$, where $\mathbf{\Sigma}$ has log space eigenvalues ranging from 1 to 100. Then, the output is generated as $y = \mathbf{M}_2\sigma(\mathbf{M}_1\mathbf{x})$, where $\mathbf{M}_1 \in \mathbb{R}^{256\times32}$, σ is the GeLU activation function, and $\mathbf{M}_2 \in \mathbb{R}^{10\times256}$, for each input data $\mathbf{x}$. $\mathbf{M}_1$ is initialized with each element drawn from $\mathcal{N}(0, \frac{1}{32})$ and $\mathbf{M}_2$ is initialized with each element drawn from $\mathcal{N}(0, \frac{1}{256})$.

The second dataset, denoted the transformed dataset, generates each sample as $\mathbf{x}_t = \mathbf{Q}\mathbf{x} + \mathbf{d}$, where $\mathbf{x}$ is a sample from the original. Here, $Q \in \mathbb{R}^{32\times32}$ is a symmetric matrix with linear space eigenvalues ranging from 1 to 100, and $\mathbf{d} \in \mathbb{R}^{32}$ is a random vector with elements drawn from the standard normal. Then, $\mathbf{y}_t = \mathbf{M}_2\sigma(\mathbf{M}_1\mathbf{x}_t)$, using the same method as in the other dataset. Both datasets are split into a $0.8 : 0.2$ into training data and validation data. This induces an affine transform of the input activations.

We then initialize 4 models, 2 to be trained on the original dataset (one optimized with SGD, and one optimized with DoPr-SGD), and 2 to be trained on the transformed dataset (one optimized with SGD, and one optimized with DoPr-SGD). The two models trained on the original dataset are initialized identically, and the two trained on the transformed dataset are also initialized identically. All models have an input layer of dimension $(32, 256)$, hidden layer $(256, 256)$, and output layer $(256, 10)$, and use SiLU activations.

For all models, the hidden layer and output layers are identical. For the original dataset models, let the input layer have weight $\mathbf{W}$, and bias $\mathbf{b}$. Then, for the transformed dataset models, the input layer is initialized with weight $\mathbf{W}\mathbf{Q}^{-1}$, and bias $\mathbf{b} - \mathbf{W}\mathbf{Q}^{-1}\mathbf{d}$.

We note here, inputting $\mathbf{x}$ into the models trained on the original data yield $\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Then, $\mathbf{x}_t = \mathbf{Q}\mathbf{x} + \mathbf{d}$, and when passed through the transformed models the output will be

$$\mathbf{h}_t = \mathbf{W}\mathbf{Q}^{-1}(\mathbf{Q}\mathbf{x} + \mathbf{d}) + \mathbf{b} - \mathbf{W}\mathbf{Q}^{-1}\mathbf{d} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

So, for all models, the pre-activations after the first layer will be identical, and thus at initialization, *all models will have identical outputs*.

From here, both models are trained for 50 epochs of 20 steps per epoch, with batch size 2048. At each step, the models are validated on the 0.2 validation set for their respective datasets, and the results reported. The models are trained so that at each training step, each batch exactly corresponds (e.g. for a batch $\mathbf{X} \in \mathbb{R}^{32\times 2048}$ of the Original data, the transformed batch will exactly be $\mathbf{X}_t = \mathbf{Q}\mathbf{X} + \mathbf{b}\mathbf{1}^T$).

For the models trained using `DoPr`, damping is set at $10^{-8} \cdot \mathrm{tr}(\mathbf{\Sigma})$, where $\mathbf{\Sigma}$ is the preconditioner.

The identical trajectories of both `DoPr` models demonstrates the affine invariance demonstrated in Proposition 4.2 – when learning the same function, and with identical initialization, affine transforms to the input distribution do not affect the loss trajectory when trained with `DoPr`. However, loss trajectories under vanilla SGD do exhibit a drift, demonstrating the coordinate-dependent trajectory as stated in Section 4.2.

C.4 Hyperparameter Scaling DoPr

This section provides details on the experiments shown in Figure 4, Figure 11, and Figure 10. The goals of these experiments are to provide guidelines for scaling hyperparameters as width increases, as discussed in Yang et al. (2022) and related work (see Appendix A). These scalings allow for zero-shot hyperparameter transfer across models with differing width.

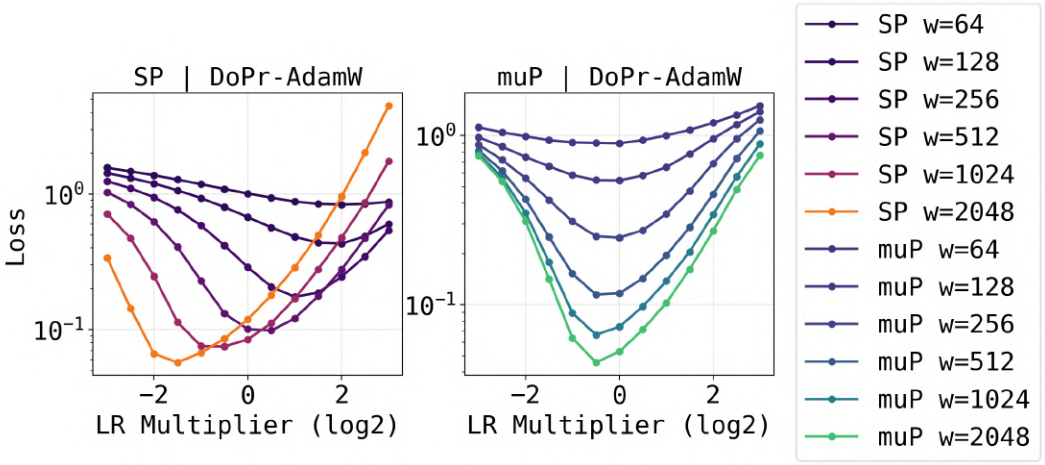


Figure 10: We train a 3-layer MLP on CIFAR, and sweep over learning rates. We compare Standard Parametrization (SP, left) and Maximal Update parametrization (μP, right). The final train loss is reported for both μP and SP. μP achieves learning rate transfer as width increases.

Models and datasets. In Figure 4 and Figure 11, for language modeling, we use a GPT2-style Transformer with 4 layers, head dimension 64, sequence length 1024, and batch size 12, trained on the OpenWebText dataset. Each experiment processes approximately 100M training tokens.

For Figure 10 and the weight-decay μP plots in Figure 4, we use 3 layer MLPs trained on CIFAR10 Krizhevsky et al., normalizing input data to be between $[-1, 1]$. The MLP layers have no bias, and for width w, the layers are of size $(3072, w)$, (w, w), and $(w, 10)$, with cross entropy loss. For the learning rate sweeps (Figure 10), we use batch size 512, and train for 8 epochs. For the weight decay experiment, we use batch size 4096 and train for 200 epochs.

Optimization. For the language modeling and CIFAR learning rate experiments we sweep over learning rate. For the language modeling we use base weight decay of 0.1, and for CIFAR we use base weight decay of 0.01. For language modeling, a base learning rate of $6 \cdot 10^{-3}$ was used. For the

Standard Parametrization (SP) sweep for CIFAR, we use base learning rate of 2^{-10}, and for the μP sweep for CIFAR, we use base learning rate of 2^{-4}. Then, the learning rate used is:

$$\eta_{base} = \eta \cdot 2^k$$

where k is the sweep multiplier. In all experiments, all layers are preconditioned, with damping that scales with $\text{tr}(\mathbf{\Sigma})$, where $\mathbf{\Sigma}$ is the preconditioner. The language model used damping $10^{-4} \cdot \text{tr}(\mathbf{\Sigma})$, and the CIFAR experiments used damping $10^{-3} \cdot \text{tr}(\mathbf{\Sigma})$.

Noting that the preconditioner $\mathbf{\Sigma}$ is the same as the preconditioner in `KFAC`, this damping aligns with the rescaled damping in Ishikawa & Karakida (2023), which naturally achieves the μP scaling with width. We also EMA the preconditioner with $\beta = 0.9$, only in the CIFAR experiments. In language modeling, we report the best training loss achieved during training. For the CIFAR learning rate sweep, we report the training loss at the final epoch for the learning rate sweep.

For the CIFAR weight decay experiment, we report $\frac{||\Delta \mathbf{W}||_F}{||\mathbf{W}||_F}$, the ratio of the update norm to the weight norm for the hidden layer $\mathbf{W}$. We report this at each step in training. We use a base learning rate of $\eta_{base} = 2^{-4}$, and a base weight decay of $\lambda_{base} = 0.01$.

μP vs SP. In all experiments, we follow the μP scaling rules as described in Yang et al. (2022). As further explained below, we note that for `DoPr-AdamW` and `DoPr-Muon` in particular, the same scaling rules apply as for `AdamW` and `Muon`. This is because of the normalizing nature of these optimizers - despite `DoPr` altering the gradient that `AdamW` and `Muon` act on, the final object still has the same normalized size.

For the μP sweeps, the input layers, are initialized as $\mathbf{W}_{ij} \sim \mathcal{N}(0, d_{in}^{-1})$. The hidden layers are initialized as $W_{ij} \sim \mathcal{N}(0, w^{-1})$, and the output layers are initialized as $\mathbf{W}_{ij} \sim \mathcal{N}(0, w^{-2})$. The input layer learning rate is η_{base}, the hidden layer learning rate is $\eta_{base} \cdot w^{-1}$, and the output layer learning rate is $\eta_{base} \cdot w^{-1}$. In μP, during attention instead of computing $\frac{q^T k}{\sqrt{d}}$, $\frac{q^T k}{d}$ is used in attention.

Given a base weight decay of $\lambda_{base} = 0.01$, the input layer's weight decay is λ_{base}, the hidden layers' weight decay is $\lambda_{base} \cdot w$, and the output layer's weight decay is $\lambda_{base} \cdot w$. This style of weight decay keeps $\eta \cdot \lambda$ width-independent has been proposed by several past work (Kosson et al., 2024; 2025; Wang & Aitchison, 2025; Bergsma et al., 2025). Other works, such as Qiu et al. (2025); Xiao (2025), propose other methods of scaling weight decay, such as scaling weight decay with $\frac{1}{w}$. We find that the scaling of weight decay with width allows the relative update size to remain invariant towards the infinite width limit.

For SP in the learning rate sweeps, all layers are set at their PyTorch defaults. In particular, the input layers are initialized at $\mathbf{W}_{ij} \sim \mathcal{N}(0, d_{in}^{-1})$. The hidden and output layers are initialized as $\mathbf{W}_{ij} \sim \mathcal{N}(0, w^{-1})$. All layers have constant learning rate, and constant weight decay.

For the weight decay experiments, for the SP run we keep the μP initialization and learning rate scaling, and use a constant weight decay for all layers.

Justification of `DoPr-AdamW` and `DoPr-Muon` μP scaling Here we provide explanation for why `DoPr` does not affect the μP scaling rules for `AdamW` and `Muon`. Intuitively, the reason for this is because `AdamW` and `Muon` can both be seen as normalizing the gradient object - so regardless of whether it is applied to the raw gradient, or the `DoPr`-gradient, the result will still have similar magnitude elements.

Let the weight in question be $\mathbf{W}$ at initialization, with raw gradient $\mathbf{G}$, and preconditioner $\mathbf{\Sigma}$. For simplicity we consider batch size of 1, let $\mathbf{W}$ be square, and let $\mathbf{z} \in \mathbb{R}^w$ be the incoming data that the optimizer updates on. We assume $\mathbf{z}$ has $\Theta(1)$ entries with respect to width. We examine the one-step update.

The preconditioned gradient is thus, for a damping constant ρ

$$\mathbf{M} = \mathbf{G}(\mathbf{\Sigma} + \rho \cdot \text{tr}(\mathbf{\Sigma})\mathbf{I})^{-1}$$

$$\mathbf{D} = \texttt{Muon}(\mathbf{M})$$

And so, the update is

$$\mathbf{W}_+ = \mathbf{W} - \eta \mathbf{D}$$

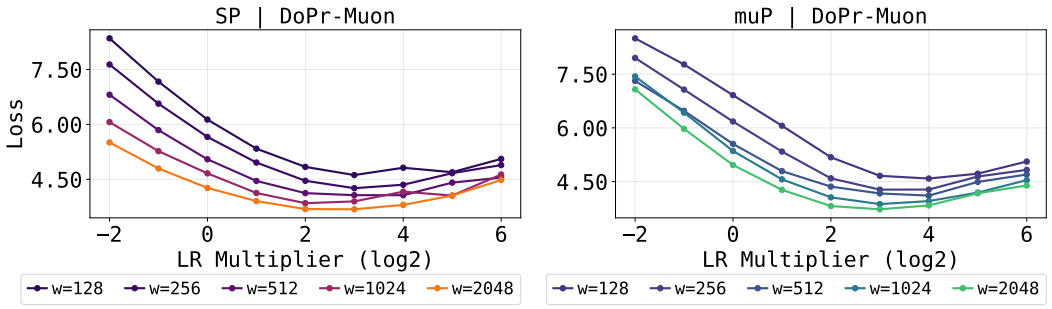


Figure 11: μP scaling behavior. Learning-rate sweeps comparing for `DoPr-Muon` under non-μP(SP) and μP parameterizations on GPT2 language model.

We note that `Muon` orthonormalizes the gradient matrix (in this case $\mathbf{M}$), up to numerical errors in the Newton-Schulz iterations. Thus, we may treat $\mathbf{D}$ as orthonormal. Then, we get:

$$\mathbf{W}_+\mathbf{z} = \mathbf{W}\mathbf{z} - \eta\mathbf{D}\mathbf{z}$$

The preconditioning only affects the $\mathbf{D}\mathbf{z}$ term, so we focus on this term. Since $\mathbf{D}$ is approximately orthonormal, and $\mathbf{z}$ has $\Theta(1)$ entries, $\mathbf{D}\mathbf{z}$ will also have $\Theta(1)$ entries, as $\mathbf{D}$ is orthonormal (for square $\mathbf{M}$). Thus, `DoPr-Muon` achieves learning rate transfer across width with constant learning rate, matching `Muon`.

In the case of `AdamW`, we consider

$$\mathbf{W}_+ = \mathbf{W} - \eta\texttt{AdamW}(\mathbf{M})$$

We consider `AdamW` without the EMA buffers, since momentum does not affect μP (Qiu et al., 2025). In this case, `AdamW` is simply an elementwise sign function. We can also note that $\mathbf{G}$ can be written as $\delta\mathbf{z}^\top$, where δ is the gradient of the loss with respect to $\mathbf{W}\mathbf{z}$. Similarly, we have $\mathbf{\Sigma} = \mathbf{z}\mathbf{z}^\top$. Thus, we see

$$\mathbf{M} = (\delta\mathbf{z}^\top)(\mathbf{z}\mathbf{z}^\top + \rho \cdot \mathrm{tr}(\mathbf{z}\mathbf{z}^\top)\mathbf{I})^{-1}$$

Thus, applying an elementwise sign function to this will create a matrix $\mathbf{D} = \texttt{AdamW}(\mathbf{M})$ with all elements $\Theta(1)$, but correlated with $\mathbf{z}$. Thus, we get

$$\mathbf{W}_+ = \mathbf{W}\mathbf{z} - \eta\mathbf{D}\mathbf{z}$$

Again, the first term is the same as in regular `AdamW`. In the second term, as $\mathbf{D}$ and $\mathbf{z}$ are correlated, by the Law of Large Numbers the resulting terms will be $\Theta(w)$, and so η should be scaled with w^{-1}, exactly matching the learning rate scaling for regular `AdamW`. Notice in this case, w is just the dimension of $\mathbf{z}$.

So, in either case, `DoPr-AdamW` and `DoPr-Muon` retains the same μP scaling rules as `AdamW` and `Muon` themselves.

C.5 State-based Imitation Learning

This section provides implementation details and supporting results for `Humanoid-v5` and `Half-Cheetah-v5` Gymnasium benchmarks (Towers et al., 2025)

Model and architecture. To maintain consistency with prior work, we use the Residual MLP architecture together with model EMA and a cosine learning rate scheduler following Block et al. (2024). Full architectural hyperparameters are reported in Table 1.

Training setup. For each optimizer variant, we conduct grid searches over learning rate (η), damping (γ), and weight decay (λ). The best-performing hyperparameter configurations are reported in Table 2 for `Humanoid-v5` and Table 3 for `Half-Cheetah-v5`. We train three independent seeds over 1000 training steps.

Evaluation Protocol. We evaluate model performance using episodic return on the corresponding control task. For each seed, we use a bootstrapped-mean evaluation procedure described in Appendix C.5 and report the 10/50/90 quantiles.

Algorithm 3 Bootstrap Mean Estimation of Success Rate

Input: checkpoints $\{c_k\}_{k=1}^{K}$, rollouts per checkpoint R, bootstrap trials B, sample size m
$\mathcal{D} \leftarrow [\,]$
for $k = 1, \ldots, K$ **do**
  Load checkpoint c_k
  for $r = 1, \ldots, R$ **do**
    Run one rollout episode
    Append outcome $x \in \{0, 1\}$ to $\mathcal{D}$ $\{1 = \text{success}\}$
  end for
end for
for $b = 1, \ldots, B$ **do**
  Draw subset $\mathcal{S}_b \subset \mathcal{D}$ of size m (without replacement)
  $\hat{\mu}_b \leftarrow \frac{1}{m} \sum_{x \in \mathcal{S}_b} x$
end for
Return: $\text{mean}(\hat{\mu}),\ q_{0.10}, q_{0.50}, q_{0.90}$

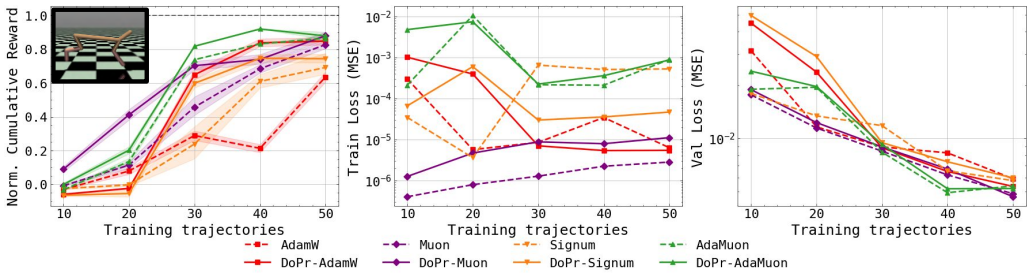


Figure 12: Half-Cheetah-v5 `DoPr` performance across `AdamW`, `Muon`, `Signum`, and `AdaMuon`. `DoPr` variants attain higher terminal reward which does not consistently correlate with train or validation loss improvements.

Results. Across both `Humanoid-v5` (Figure 5) and `Half-Cheetah-v5` (Figure 12), `DoPr` variants consistently improve reward relative to their baseline optimizer counterparts. These gains do not reliably track training or validation loss trends.

Table 1: State-based IL Network Architecture and Training Configuration

Parameter	Value
Hidden layers	[256, 256]
Activation	ReLU
Batch size	256
Training steps	1000
Eval trajectories	100
LR scheduler	Cosine annealing
Warmup fraction	0.05
Min LR (η_{min})	0.0
Model EMA	0.99

C.6 Image-Based Robot Policy Learning

This section provides additional implementation details and supporting results for the `Tool-Hang` and `Transport` Robomimic tasks (Mandlekar et al., 2021)

Model and architecture. We evaluate `DoPr` performance on state-of-the-art imitation learning architectures with generative control policies (GCPs) using the image-based `Tool Hang` and `Transport` tasks at Proficient Human (PH) expert level from Robomimic (Mandlekar et al., 2021). We use the training pipeline of Pan et al. (2025), inheriting all architectural and environment

Table 2: Best optimizer hyperparameters across trajectory counts for Humanoid.

Method	10 traj	20 traj	30 traj	40 traj	50 traj
AdamW	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$	$\eta = 3\times10^{-3}$ $\lambda = 0$	$\eta = 3\times10^{-3}$ $\lambda = 0$	$\eta = 1\times10^{-3}$ $\lambda = 0$
DoPr-AdamW	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-4}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-4}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-5}$
Muon	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 2\times10^{-2}$ $\lambda = 5\times10^{-3}$	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 2\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 1\times10^{-2}$ $\lambda = 5\times10^{-3}$
DoPr-Muon	$\eta = 2\times10^{-2}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-4}$	$\eta = 2\times10^{-2}$ $\lambda = 5\times10^{-3}$ $\gamma = 1\times10^{-4}$	$\eta = 2\times10^{-2}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-4}$	$\eta = 2\times10^{-2}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-5}$	$\eta = 2\times10^{-2}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-5}$
Signum	$\eta = 3\times10^{-4}$ $\lambda = 0$	$\eta = 1\times10^{-3}$ $\lambda = 0$	$\eta = 1\times10^{-3}$ $\lambda = 0$	$\eta = 3\times10^{-4}$ $\lambda = 0$	$\eta = 3\times10^{-4}$ $\lambda = 0$
DoPr-Signum	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$	$\eta = 3\times10^{-4}$ $\lambda = 0$ $\gamma = 1\times10^{-4}$	$\eta = 3\times10^{-4}$ $\lambda = 0$ $\gamma = 1\times10^{-5}$	$\eta = 3\times10^{-4}$ $\lambda = 0$ $\gamma = 1\times10^{-4}$	$\eta = 3\times10^{-4}$ $\lambda = 0$ $\gamma = 1\times10^{-5}$
AdaMuon	$\eta = 1\times10^{-3}$ $\lambda = 1\times10^{-1}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$	$\eta = 1\times10^{-3}$ $\lambda = 1\times10^{-2}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-1}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-1}$
DoPr-AdaMuon	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-2}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-1}$ $\gamma = 1\times10^{-2}$	$\eta = 1\times10^{-3}$ $\lambda = 1\times10^{-1}$ $\gamma = 1\times10^{-4}$	$\eta = 1\times10^{-3}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-4}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-3}$

Table 3: Best optimizer hyperparameters across trajectory counts for HalfCheetah.

Method	10 traj	20 traj	30 traj	40 traj	50 traj
AdamW	$\eta = 2\times10^{-3}$ $\lambda = 3\times10^{-3}$	$\eta = 2\times10^{-4}$ $\lambda = 0$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$	$\eta = 2\times10^{-3}$ $\lambda = 3\times10^{-2}$	$\eta = 2\times10^{-3}$ $\lambda = 3\times10^{-2}$
DoPr-AdamW	$\eta = 1\times10^{-4}$ $\lambda = 0$ $\gamma = 1\times10^{-5}$	$\eta = 3\times10^{-4}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$	$\eta = 2\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$	$\eta = 2\times10^{-3}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-6}$	$\eta = 2\times10^{-3}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-3}$
Muon	$\eta = 2\times10^{-2}$ $\lambda = 5\times10^{-3}$	$\eta = 2\times10^{-2}$ $\lambda = 5\times10^{-3}$	$\eta = 3\times10^{-2}$ $\lambda = 5\times10^{-3}$	$\eta = 3\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 3\times10^{-2}$ $\lambda = 2\times10^{-2}$
DoPr-Muon	$\eta = 5\times10^{-3}$ $\lambda = 5\times10^{-3}$ $\gamma = 3\times10^{-4}$	$\eta = 3\times10^{-2}$ $\lambda = 2\times10^{-2}$ $\gamma = 1\times10^{-3}$	$\eta = 3\times10^{-2}$ $\lambda = 2\times10^{-2}$ $\gamma = 1\times10^{-3}$	$\eta = 3\times10^{-2}$ $\lambda = 5\times10^{-3}$ $\gamma = 1\times10^{-3}$	$\eta = 3\times10^{-2}$ $\lambda = 2\times10^{-2}$ $\gamma = 1\times10^{-3}$
Signum	$\eta = 1\times10^{-3}$ $\lambda = 0$	$\eta = 3\times10^{-4}$ $\lambda = 0$	$\eta = 3\times10^{-3}$ $\lambda = 0$	$\eta = 3\times10^{-3}$ $\lambda = 0$	$\eta = 3\times10^{-3}$ $\lambda = 0$
DoPr-Signum	$\eta = 1\times10^{-4}$ $\lambda = 0$ $\gamma = 1\times10^{-4}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-5}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$	$\eta = 1\times10^{-3}$ $\lambda = 0$ $\gamma = 1\times10^{-3}$
AdaMuon	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$
DoPr-AdaMuon	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-3}$	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-2}$	$\eta = 1\times10^{-3}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-3}$	$\eta = 1\times10^{-2}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-2}$	$\eta = 3\times10^{-3}$ $\lambda = 1\times10^{-2}$ $\gamma = 1\times10^{-3}$

hyperparameters. Our primary policy architecture is ChiUNet (Chi et al., 2023), with approximately 20M parameters.

Training setup. We conduct learning rate sweeps over `AdamW`, `Muon`, and their corresponding `DoPr` variants. We precondition all layers (including FiLM) with full preconditioning (via Cholesky) except for Conv2d layers which use the SUA approximation (Appendix B.3.2). For each method,

Table 4: Best optimizer hyperparameters for Tool Hang PH Image using ChiUNet (seeds 42, 43, 44).

Hyperparameter	AdamW	DoPr-AdamW	Muon	DoPr-Muon
lr	1e−4	1e−4	3e−4	3e−4
weight_decay	1e−5	1e−5	0.1	0.1
muon_momentum	—	—	0.95	0.95
grad_clip_norm	10.0	10.0	1.0	1.0
gradient_steps	300k	300k	200k	200k
batch_size	256	256	256	256
precond_ema_rate	—	0.9	—	0.9
grad_ema_rate	—	0.9	—	0.9
precond_include_film	—	true	—	true
damping	—	1e−4	—	1e−4
damping_mode	—	trace	—	trace

Table 5: Best optimizer hyperparameters for Transport MH Image using ChiUNet (seeds 42, 43, 44).

Hyperparameter	AdamW	DoPr-AdamW	Muon	DoPr-Muon
lr	1e−4	1e−4	1e−4	1e−4
weight_decay	1e−5	1e−5	0.1	0.1
muon_momentum	—	—	0.95	0.95
grad_clip_norm	10.0	10.0	1.0	1.0
gradient_steps	300k	300k	200k	200k
batch_size	256	256	256	256
precond_ema_rate	—	0.0	—	0.0
grad_ema_rate	—	0.0	—	0.0
precond_include_film	—	true	—	true
damping	—	1e−4	—	1e−4
damping_mode	—	trace	—	trace

we tune learning rates while keeping the remaining training configuration fixed to the reference implementation. Full optimizer hyperparameters are reported in Table 4 and Table 5.

Evaluation Protocol. Policies are evaluated using task success rate under the standard Robomimic benchmark protocol. During training, we evaluate checkpoints every 20k steps using 9-step action sampling and report the corresponding rollout success rate using the bootstrapped mean procedure Appendix C.5.

Results. Results are summarized in Figure 6. Across both tasks, DoPr variants consistently match or exceed their baseline counterparts, demonstrating that the proposed preconditioning strategy transfers effectively to high-dimensional visuomotor control.

C.7 Supervised Fine-Tuning

This section provides full details for the supervised fine-tuning (SFT) experiments used in Section 5.3. We report two SFT settings. The first is a small-scale Llama-3.2-3B GSM8K sweep used to characterize learning-rate sensitivity, training-time sample efficiency, and the relationship between downstream accuracy and token-level training loss. The second is our primary large-scale math SFT experiment, which fine-tunes Llama-3.1-8B with LoRA on the OpenMathInstruct-2 train_1M split and evaluates on GSM8K, GSM8K-CoT, and MATH-500.

C.7.1 3B SFT

Overview. This experiment uses the Llama-3.2-3B base model AI@Meta (2024); Grattafiori et al. (2024) with LoRA Hu et al. (2022) on a 100K-sample subset of OpenMathInstruct-2 Toshniwal et al. (2024) for one epoch.

Additional results We report the Muon SFT sweep results in Figure 15. Consistent with the DoPr-AdamW comparison, DoPr-Muon also achieves stronger downstream GSM8K performance

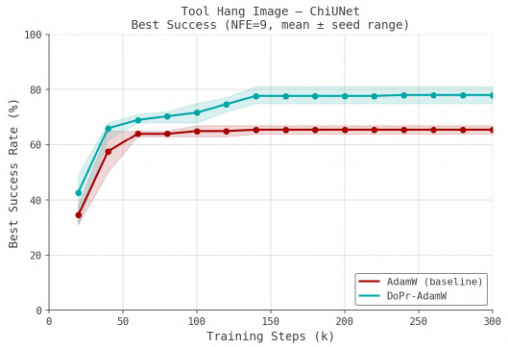


Figure 13a: Tool Hang: AdamW vs `DoPr`-AdamW

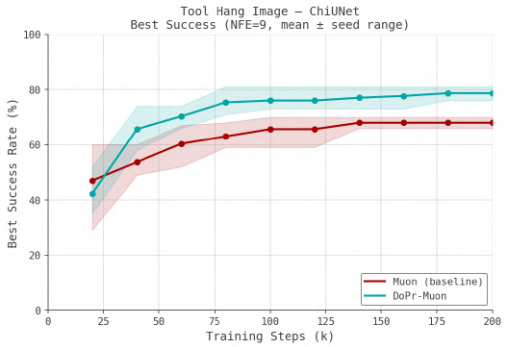


Figure 13b: Tool Hang: Muon vs `DoPr`-Muon

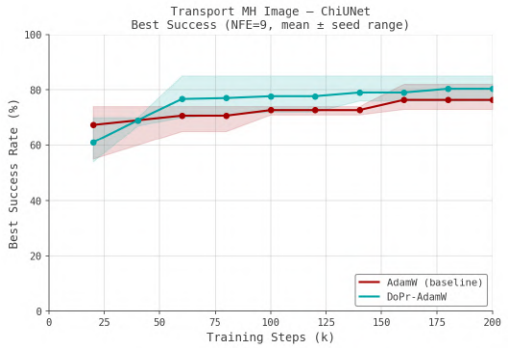


Figure 13c: Transport: AdamW vs `DoPr`-AdamW

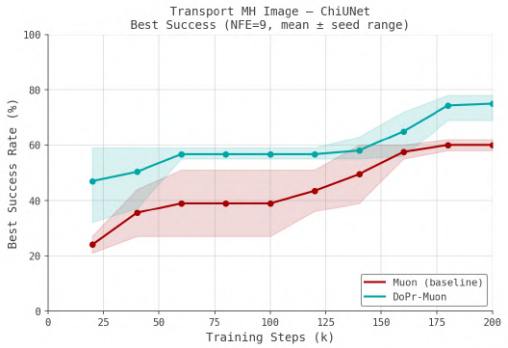


Figure 13d: Transport: Muon vs `DoPr`-Muon

Figure 13: Best mean success rate throughout training on `Tool-Hang` and `Transport` tasks from `robomimic`. We compare AdamW, Muon, and their `DoPr` variants, reporting the best mean success rate over evaluation checkpoints every 20k steps.

across the learning rate sweep, while often demonstrating higher final training loss than the base `Muon` optimizer. This further supports our main observation that training loss alone can be a poor proxy for downstream performance, and that `DoPr` improves the optimization in a way that is better aligned with downstream generalization.

Model and architecture. We fine-tune the `Llama-3.2-3B` base model AI@Meta (2024); Grattafiori et al. (2024) using the `Llama-3.2-3B-Instruct` tokenizer. All experiments use bfloat16 precision for weights and activations, and implement attention with FlashAttention-2.

Dataset and preprocessing. Training data are drawn from OpenMathInstruct-2 Toshniwal et al. (2024), using a fixed 100K-example subset. We tokenize sequences to a maximum context length of 1024 tokens and enable sequence packing for efficiency.

Parameter-efficient fine-tuning. We apply LoRA Hu et al. (2022) to all linear layers with rank $r = 16$ and scale $\alpha = 16$. We additionally train the token embedding matrix to accommodate chat-template special tokens, while preserving weight tying between the input embeddings and output projection.

Optimization and training. We train for one epoch using either `AdamW` or `Muon`, each with and without the `DoPr` wrapper. The base learning rate is 1×10^{-5}, with cosine scheduling and 5% warmup. We use per-device batch size 2 and gradient accumulation over 8 steps.

Evaluation protocol. Downstream evaluation uses `lm-eval` with the `vLLM` backend in bfloat16 precision. We evaluate on `GSM8K` with 5-shot prompting, enabling chat-style prompting via the model's native template and formatting few-shot exemplars as multi-turn conversations.

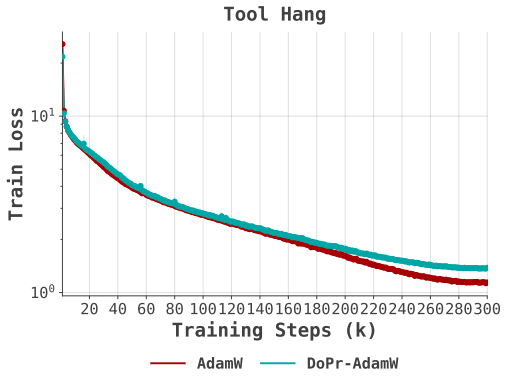


Figure 14a: Tool Hang: AdamW vs `DoPr`-AdamW

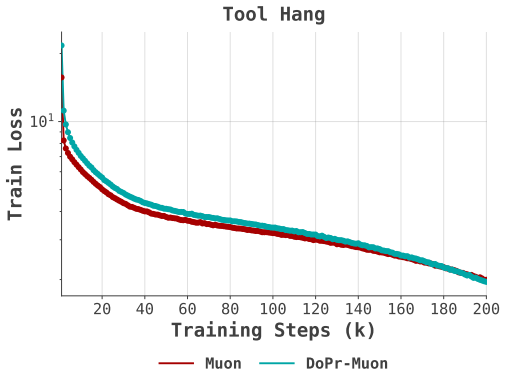


Figure 14b: Tool Hang: Muon vs `DoPr`-Muon

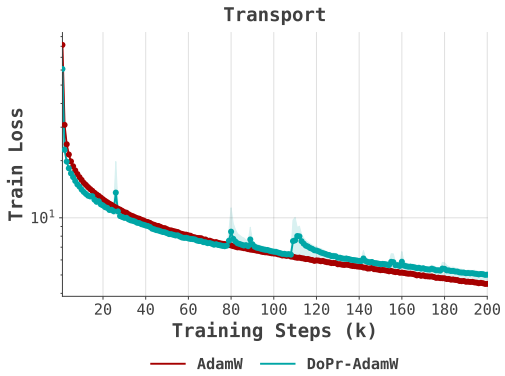


Figure 14c: Transport: AdamW vs `DoPr`-AdamW

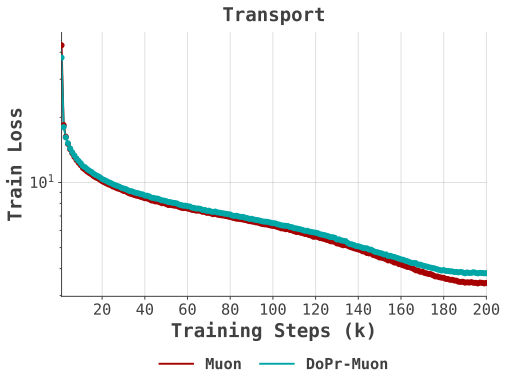


Figure 14d: Transport: Muon vs `DoPr`-Muon

Figure 14: Training Loss on `Tool-Hang` and `Transport` tasks from `robomimic`. We compare AdamW, Muon, and their `DoPr` variants

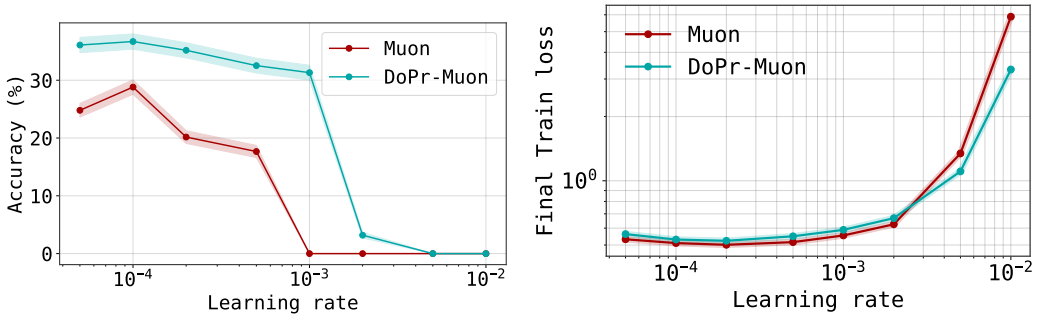


Figure 15: `GSM8K` **3B SFT sweep**. Peak `GSM8K` accuracy across training steps vs. learning rate, comparing between `Muon` and `DoPr-Muon`.

C.7.2 8B SFT

Overview. Our primary SFT experiment uses `meta-llama/Llama-3.1-8B` checkpoint AI@Meta (2024); Grattafiori et al. (2024) as the frozen base model. We train only LoRA adapters Hu et al. (2022); the base model weights remain frozen throughout training. The tokenizer and chat template are taken from `meta-llama/Llama-3.1-8B-Instruct`. Training data are drawn from `nvidia/OpenMathInstruct-2` Toshniwal et al. (2024), split `train_1M`. The primary comparison in this setting is between `AdamW` and `DoPr-AdamW`, where `DoPr-AdamW` applies `AP` to the trainable LoRA modules before the AdamW update.

Table 6: Configuration for the small-scale GSM8K SFT sweep.

Component	Setting
Base model	`Llama-3.2-3B` AI@Meta (2024); Grattafiori et al. (2024)
Tokenizer	`Llama-3.2-3B-Instruct` tokenizer
Precision	bfloat16 weights and activations
Attention implementation	FlashAttention-2 Dao et al. (2022); Dao (2024)
Training dataset	OpenMathInstruct-2 Toshniwal et al. (2024)
Data subset	100K examples sampled from the 1M split
Context length	1024 tokens
Packing	Enabled
PEFT method	LoRA Hu et al. (2022)
LoRA target modules	All linear layers
LoRA rank / scale	$r = 16$, $\alpha = 16$
Additional trainables	Token embedding matrix, used for chat-template special tokens
Weight tying	Preserved between input embeddings and output projections
Optimizers compared	`AdamW` and `Muon`, each with and without the `DoPr` wrapper
Training length	1 epoch
Base learning rate	1×10^{-5}
LR schedule	Cosine decay with 5% warmup
Per-device batch size	2
Gradient accumulation	8 steps
Eval framework	`lm-eval` with `vLLM` backend
Eval precision	bfloat16
Benchmark	`GSM8K`, 5-shot
Prompting format	Chat template enabled; few-shot examples are formatted as multi-turn conversations

Table 7: 8B SFT sweep. Final accuracy is reported as percent $\pm$ one standard error for `AdamW` and `DoPr-AdamW` on `GSM8K`, `GSM8K-CoT`, and `MATH-500`. Δ denotes the improvement of `DoPr-AdamW` over `AdamW` in percentage points.

Learning rate	GSM8K			GSM8K-CoT			MATH-500		
	`AdamW`	`DoPr-AdamW`	Δ	`AdamW`	`DoPr-AdamW`	Δ	`AdamW`	`DoPr-AdamW`	Δ
2×10^{-5}	71.4±1.2	**71.9**±1.2	+0.5	**72.2**±1.2	71.9±1.2	−0.2	33.8±2.1	33.8±2.1	0.0
5×10^{-5}	68.1±1.3	**70.1**±1.3	+2.0	69.4±1.3	**71.6**±1.2	+2.1	34.4±2.1	**35.0**±2.1	+0.6
7×10^{-5}	**69.7**±1.3	69.5±1.3	−0.2	69.9±1.3	**70.1**±1.3	+0.2	29.6±2.0	**34.4**±2.1	+4.8
1×10^{-4}	**68.1**±1.3	66.7±1.3	−1.4	**72.5**±1.2	68.5±1.3	−3.9	**33.0**±2.1	32.2±2.1	−0.8
2×10^{-4}	78.0±1.1	**80.1**±1.1	+2.1	**81.0**±1.1	75.7±1.2	−5.2	**41.6**±2.2	38.6±2.2	−3.0
5×10^{-4}	66.2±1.3	**80.5**±1.1	+14.3	51.6±1.4	**81.9**±1.1	+30.3	46.4±2.2	**47.4**±2.2	+1.0
7×10^{-4}	56.0±1.4	**70.4**±1.3	+14.4	14.5±1.0	**78.4**±1.1	+63.9	42.6±2.2	**45.8**±2.2	+3.2
1×10^{-3}	**2.9**±0.5	2.2±0.4	−0.7	**2.8**±0.5	2.0±0.4	−0.8	3.6±0.8	**4.6**±0.9	+1.0
Mean Δ over LR grid			+3.9			+10.8			+0.9

Additional results. Results are reported in Table 7. Across the full learning-rate grid, `DoPr-AdamW` improves the average final accuracy over `AdamW` by $+3.9$, $+10.8$, and $+0.9$ percentage points on `GSM8K`, `GSM8K-CoT`, and `MATH-500`, respectively. The gains are especially pronounced in the high-learning-rate regime: at 5×10^{-4}, `DoPr-AdamW` improves `GSM8K-CoT` from 51.6% to 81.9%, and at 7×10^{-4}, from 14.5% to 78.4%. Thus, beyond improving the best observed endpoint performance, `DoPr` substantially broadens the stable and effective learning-rate range for large-scale SFT. Taking the best final checkpoint over the sweep, `DoPr-AdamW` reaches 80.5% on `GSM8K`, 81.9% on `GSM8K-CoT`, and 47.4% on `MATH-500`, improving over the best `AdamW` result on all three benchmarks.

Training data and prompt construction. Each training example consists of a math problem and a generated solution from OpenMathInstruct-2. The `problem` field is inserted into the fixed user prompt:

Table 8: Core configuration for the large-scale 8B SFT experiment.

Component	Setting
Base model	`meta-llama/Llama-3.1-8B`
Tokenizer / chat template	`meta-llama/Llama-3.1-8B-Instruct`
Precision	bfloat16 model and compute; float32 preconditioner statistics
Attention implementation	FlashAttention-2 Dao et al. (2022); Dao (2024)
Trainer	`trl.SFTTrainer` with PEFT LoRA
Training dataset	`nvidia/OpenMathInstruct-2`, split `train_1M`
Input fields	`problem` and `generated_solution`
Objective	Completion-only next-token loss; user-prompt tokens are masked
Maximum sequence length	4096 tokens
Packing	Enabled
Training length	2 epochs
Processed tokens	997,968,955 tokens
Optimizer steps	482
Effective global batch size	512 packed sequences
World size	8 GPUs
Per-device micro-batch size	2
Gradient accumulation	32 steps
DeepSpeed configuration	ZeRO-3
Gradient checkpointing	Enabled, reentrant checkpointing path

Table 9: LoRA, optimizer, learning-rate sweep, and preconditioner configuration for the large-scale 8B SFT experiment.

Component	Setting
PEFT method	LoRA Hu et al. (2022)
LoRA target modules	`q,k,v,o,gate,up,down` projection matrices
LoRA rank / scale	$r = 64$, $\alpha = 128$
LoRA dropout	0.05
LoRA bias	`none`
Trainable adapter parameters	167,772,160
AdamW betas and epsilon	$(0.9, 0.98)$, $\epsilon = 10^{-8}$
Weight decay	0.01
LR schedule	Constant, no warmup

```
Solve the following math problem.  Make sure to put the
answer (and only answer) inside \boxed{}.
```

The target completion is the corresponding `generated_solution`.

Parameter-efficient fine-tuning. All large-scale SFT runs use LoRA Hu et al. (2022); full fine-tuning is not used. LoRA adapters are applied to the standard Llama attention and MLP projection matrices:

$$\{\texttt{q_proj}, \texttt{k_proj}, \texttt{v_proj}, \texttt{o_proj}, \texttt{gate_proj}, \texttt{up_proj}, \texttt{down_proj}\}.$$

The LoRA rank is $r = 64$, the LoRA scale is $\alpha = 128$, LoRA dropout is 0.05, and no bias terms are trained.

Evaluation protocol. Evaluation is performed with `lm-evaluation-harness` using the `vLLM` backend. Fine-tuned models are evaluated by loading the frozen `Llama-3.1-8B` base weights together with the LoRA adapter and the saved adapter tokenizer when present. The evaluation wrapper applies the chat template for fine-tuned adapters. Inference uses bfloat16 precision, tensor parallelism over 8 GPUs, greedy decoding, temperature 0.0, and top-$p = 1.0$. For the learning-rate sweep evaluation, we use `max_model_len=8193`, `max_gen_toks=2048`, `gpu_memory_utilization=0.9`, `batch_size=auto`, and `max_batch_size=64`. The one-token context margin is introduced to avoid vLLM scoring failures at exactly the model length.

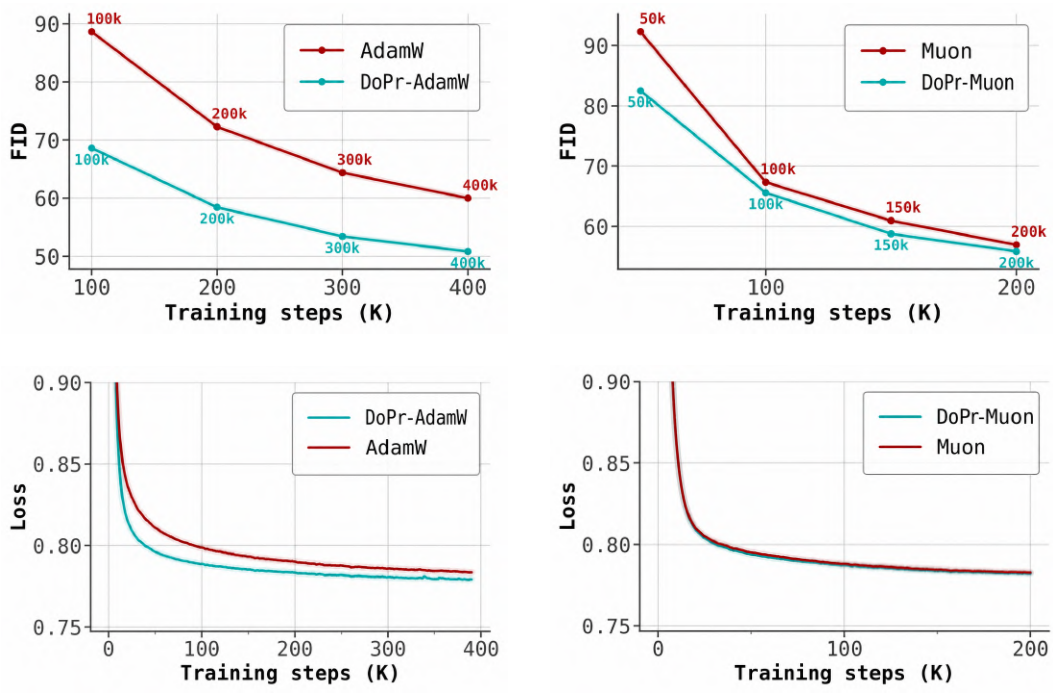


Figure 16: SiT-S on ImageNet256. **Top row**: perceptual quality (FID-50K) vs. training steps. **Bottom row:** training loss trajectories. We see that `DoPr` improves sample quality as well as loss convergence.

We report the following evaluation tasks:

- `gsm8k`: 5-shot direct `GSM8K` generation, exact match with the flexible extraction filter.
- `gsm8k_cot`: 8-shot chain-of-thought `GSM8K` generation, exact match with the flexible extraction filter.
- `minerva_math500`: 4-shot Minerva-style MATH-500 generation, scored with `math_verify`.

Reported uncertainty values are the standard errors emitted by `lm-evaluation-harness`.

C.8 Generative Modeling

Generative modeling as TTF. Generative modeling is a natural TTF setting: training is local teacher-forced velocity regression, whereas deployment rolls out the learned velocity field under its own induced trajectory. As in flow matching, the model minimizes errors on $(\mathbf{x}_t, \mathbf{y}_t)$, where $\mathbf{y}_t = \mathbf{v}(\mathbf{x}_t, t)$ is the closed-form target velocity. At generation time, however, samples are produced by iteratively integrating the learned field, e.g. $\frac{\mathrm{d}}{\mathrm{d}t}\widehat{\mathbf{z}}_t = \widehat{\mathbf{v}}(\widehat{\mathbf{x}}_t, t)$, so velocity errors can accumulate and shift future states away from the training distribution. Consequently, per-step training loss can misalign with final sample quality metrics such as FID.

We evaluate `DoPr` on generative modeling using the SiT-S model (Ma et al., 2024) trained at 256×256 resolution on ImageNet (Deng et al., 2009). As in Ma et al. (2024), the baseline model is trained with `AdamW`, and generates using a number of function evaluations (NFE) fixed to 250. We follow our general operating guidelines and swap the optimizer to `DoPr-AdamW`, maintaining the same learning rate and all other architectural, training, and evaluation protocols as the baseline. We then repeat this using baseline `Muon` and `DoPr-Muon`. Table 10 summarizes the model, training, and evaluation configuration.

We plot the FID-50K scores against training steps as the "downstream metric"; refer to Figure 16. Across both `AdamW` and `Muon`, `DoPr` consistently accelerates convergence and improves final FID. In addition, qualitative inspection of intermediate generations in Figure 17 indicates that `DoPr` induces a distinct optimization trajectory compared to the base optimizers.

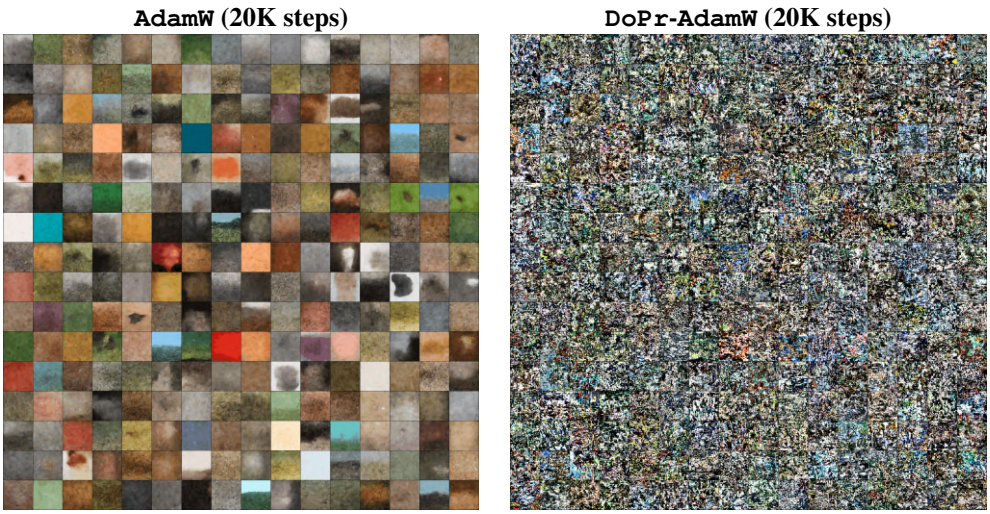


Figure 17: Generated ImageNet256 samples at 20K training steps (SiT-S, `AdamW`). Although both models are trained under identical settings and evaluated using the same ODE sampler, the samples generated by `DoPr-AdamW` exhibit qualitatively different structure and composition compared to `AdamW` at the same early training stage.

Table 10: Experimental setup for SiT-S generative modeling on ImageNet at 256×256. Across all runs, we vary only the optimizer choice and whether `DoPr` is enabled; all other training and evaluation settings match the standard SiT protocol.

Component	Setting
Model	SiT-S Ma et al. (2024)
Dataset	ImageNet Deng et al. (2009)
Resolution	256×256
Architecture / parameterization / conditioning	Exactly follows the original SiT implementation Ma et al. (2024)
Optimizers compared	`AdamW`; `Muon`; each *with* and *without* the `DoPr` wrapper
Training steps	`AdamW`: 400K steps; `Muon`: 200K steps
Learning rate	`AdamW`: matches SiT defaults Ma et al. (2024); `Muon`: 10^{-3} (per prior recommendations)
Other training settings	Held fixed across runs (data preprocessing, batch size, LR schedule, regularization)
Sampler	ODE-based sampler from SiT Ma et al. (2024)
NFE	250
Metric	FID-50K (unless otherwise specified)
FID reference set	ImageNet training set Deng et al. (2009)

Notably, as seen in Figure 16, the relative improvements in FID are somewhat proportional to *improvements* in train loss convergence. We hypothesize that generative modeling is a relatively unique TTF setting, in that one has control over the data-generating dynamical system. For example, many works study "straightening" the transport trajectory (cf. standard flow matching versus Rectified Flow (Liu et al., 2023) and Optimal-Transport CFM (Tong et al., 2024)), which can enable low NFE sampling. Therefore, the severity of TTF is tied to the generative algorithm. We leave exploring this space for co-design to future work.

D Additional Experiments and Setup

D.1 Moment Grafting Experiments

Datasets. The "original" dataset used is CIFAR10 Krizhevsky et al.. We split this into a 0.8:0.2 split of train/validation data. Then, we generate a "transformed" dataset by applying a random spatial

rotation of CIFAR by $90°$, and after normalizing all coordinates to be between $[-1, 1]$, we apply $\tanh$ elementwise. We note this induces a nonlinear transform.

Training. Next, we train 3 models, each a 3 layer MLP with layer dimensions $(3072, 256)$, $(256, 256)$, and $(256, 10)$, and SiLU activations. All models are initialized identically. One model is trained on the original CIFAR dataset Krizhevsky et al., called "Original." Another model is trained on the transformed CIFAR data, denoted "Transformed." We also coordinate training so that at each step, the batches the Transformed model sees corresponds exactly to the transformed version of the batches the Original model sees. The last model is called "Grafted." All models are trained simultaneously.

The Grafted model is trained as follows: during training, the activations inputted into each layer of the Transformed model are recorded, and their means/covariances calculated. Then, the Grafted model takes as input the Original data batch. Then, before passing the data through each layer's weights, an affine transform is applied to the incoming activations to match their mean and covariance to the cached means/covariances of the Transformed model's forward pass.

Concretely, let $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$ be the mean and covariance of transformed batch, $\mathbf{X}_t \in \mathbb{R}^{3072 \times b}$, where b is the batch size. When the corresponding batch $\mathbf{X}$ is passed into the Grafted model, let $\mathbf{X}$ have mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. Then, before we pass $\mathbf{X}$ through the layer, we apply the following transform to $\mathbf{X}$:

$$\mathbf{X}_{new} = \boldsymbol{\Sigma}_t^{1/2}\boldsymbol{\Sigma}^{-1/2}(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top) + \boldsymbol{\mu}_t\mathbf{1}^\top$$

We can note that as $\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top$ has mean 0, then $\mathbf{X}_{new}$ will have mean $\boldsymbol{\mu}_t$. Similarly, we can find the covariance as:

$$(\mathbf{X}_{new} - \boldsymbol{\mu}_t\mathbf{1}^\top)(\mathbf{X}_{new} - \boldsymbol{\mu}_t\mathbf{1}^\top)^\top = \boldsymbol{\Sigma}_t^{1/2}\boldsymbol{\Sigma}^{-1/2}(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top)(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^\top)^\top(\boldsymbol{\Sigma}^{-1/2})^\top(\boldsymbol{\Sigma}_t^{1/2})^\top$$

Noting that covariances matrices are symmetric:

$$= \boldsymbol{\Sigma}_t^{1/2}\boldsymbol{\Sigma}^{-1/2}\boldsymbol{\Sigma}(\boldsymbol{\Sigma}^{-1/2})^\top(\boldsymbol{\Sigma}_t^{1/2})^\top = \boldsymbol{\Sigma}_t^{1/2}\boldsymbol{\Sigma}_t^{1/2} = \boldsymbol{\Sigma}_t$$

Therefore, this transform does give $\mathbf{X}_{new}$ a mean of $\boldsymbol{\mu}_t$, and covariance of $\boldsymbol{\Sigma}_t$. However, since a nonlinear transform was used to create the transformed data, this affine transform *will not be an exact recovery of the transformed data*, only matching the mean and covariance. We repeat this process for the hidden layer and output layer, with the respective activations for each.

In these experiments, we use a batch size of 2048, and μP scaling. The base learning rate was 10^{-3} for `SGD`, 10^{-5} for `AdamW`, and 10^{-4} for `Muon`. Otherwise, the default PyTorch parameters are used for `AdamW` and `Muon`. To compute square roots and inverse square roots of the covariance matrices, symmetric eigenvalue decomposition was used, with a damping constant of $10^{-6} \cdot \mathrm{tr}(\mathbf{A})$, where $\mathbf{A}$ is the matrix to be decomposed.

Evaluation. Finally, the models are validated on the Transformed data after each training step, and the validation losses plotted. The results are displayed in Figure 18.

Despite not being trained on the Transformed data, the Grafted model's validation performance on the transformed data matches much more closely with the Transformed model than the Original model (trained on the original data) across optimizers, suggesting that when trained on data with similar means and covariances, models often learn similarly, even if the data itself does not perfectly match. While not displayed, we further note that increasing batch size and decreasing learning rate both help to improve how closely the Grafted model fits the Transformed model in validation loss, as they reduce drift arising from mean/covariance estimation noise and higher-order mismatches from the *non-linear* data transform. Notably, we remark that since there is no corrective mechanism to port the Grafted model's weights onto the Transformed model's "ground truth" weights, the drift will monotonically compound over the training horizon.

The Grafted model matching the Transformed model in validation demonstrates that even ignoring the higher-order nonlinear aspects of the forward pass, the mean and covariance of a forward pass are sufficient to *graft* much of the downstream performance of a model that is truly trained on the transformed data onto another model, despite the latter only seeing the original data in its forward/backward pass. This lends credence to the premise of `AP`, where the working hypothesis is that "error directions" (as captured by the mean and covariance of each layer's activations) strongly

bias the optimizer's downstream behavior. Grafting demonstrates that, in the hypothetical case where we *knew* the feature directions of the "downstream" model, we can graft its bias onto the training on the training distribution. However, since we typically cannot know these features ahead of time, AP—which essentially treats all feature directions equally — is the "worst-case" optimal approach.

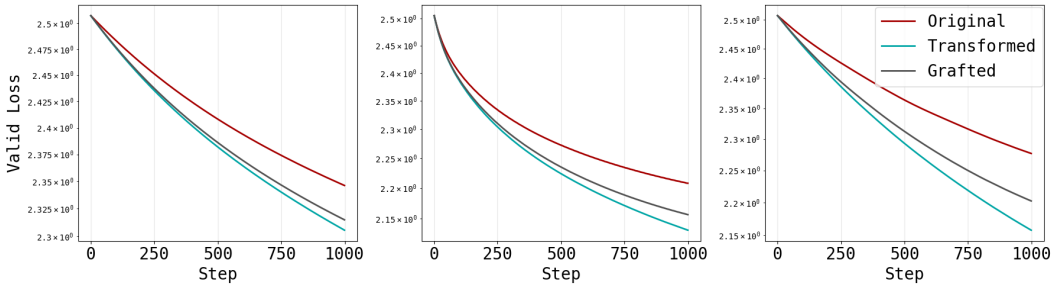


Figure 18: We train 3 MLPs on CIFAR, and a nonlinear invertible transform of CIFAR. We plot the validation loss for the 3 models for each of AdamW (left), SGD (middle), and Muon (right). Across all 3, it can be seen that the Grafted model's trajectory is closer to the Transformed model than the Original model.

D.2 Stable Ranks of Optimizer and Model Quantities

Following prior work that hypothesizes the superior training efficiency of Muon (compared to, e.g., SGD) due to ratios of certain model/optimizer quantities, we plot these relevant quantities using the terminal weights of the small-scale 3B SFT run in Section 5.3 and Appendix C.7. In particular, we follow the predictions in Davis & Drusvyatskiy (2025) and compare the ratio of the "nuclear rank" of the layer-wise gradients and the stable rank of the batch activation covariances of a forward pass, see Figure 19a. However, we note that DoPr-Muon does not orthogonalize the raw gradient, rather, it takes the AP gradient as input. Following the hypothesis in Davis & Drusvyatskiy (2025), we also compare the nuclear rank of the AP gradient to the stable rank of the activation covariances (Figure 19b).

On the other hand, we plot the standalone stable ranks of the weight matrices themselves (Figure 19c) and the batch activation covariances (Figure 19d). In this case, we do not see a consistent trend across layers, which supports our working hypothesis that our intervention on the optimizer is unrelated to regularizing the optimizer dynamics in *weight geometry*, but rather targets the *error geometry*, captured by the correspondence between the (AP) gradient and the activation.

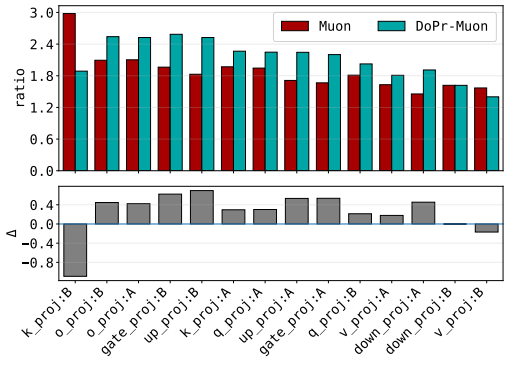


Figure 19a

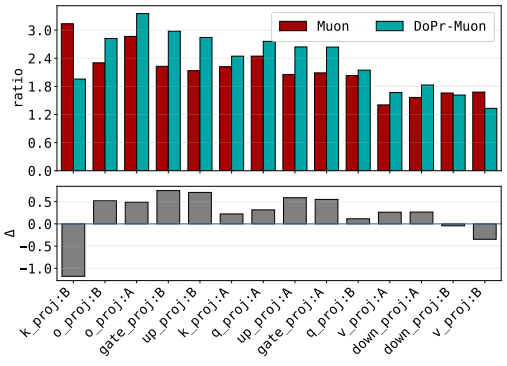


Figure 19b

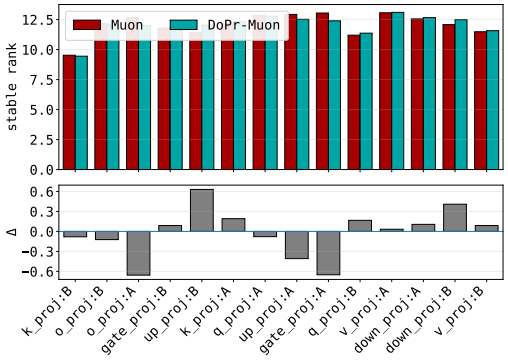


Figure 19c

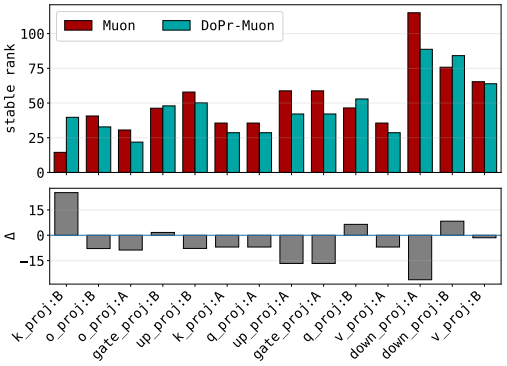


Figure 19d

Figure 19: Metric values aggregated by layer group for `Muon` and `DoPr-Muon`. The bottom panel shows the mean difference (`DoPr-Muon` minus `Muon`).