EFFICIENT LONG-HORIZON LEARNING FOR LEARNED OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Learned optimization aims to improve upon hand-designed optimizers (e.g., Adam and Muon) by meta-learning small neural network optimizers over a distribution of tasks. While recent work has greatly advanced the architectural design and inductive biases of learned optimizers (LOs), their meta-training remains biased toward short-unroll learning on particular tasks, resulting in redundant computation and leaving LOs often unable to compete with hand-designed optimizers. We introduce Efficient Long-hOrizon (ELO) learning, an efficient meta-training algorithm that (1) reallocates wasted meta-training compute to longer failure regimes, achieving efficient long-horizon learning, and (2) enforces decoupled progressive expert supervision, providing stable meta-learning signals that additionally improve the generalization of LOs. Our empirical study evaluates ELO for meta-training both element-wise and matrix-based LOs. Across downstream language modeling (GPT-2 and GPT-3 on FineWeb) and image classification (ViT-B/16, ResNet-50 on ImageNet-1K) tasks, ELO substantially improves the long-unroll performance and out-of-distribution generalization of the base LOs. In particular, ELO-Celo2 outperforms well-tuned AdamW and Muon on GPT-2 (30M and 124M) language pretraining. *Notably, all ELO baselines require less than 7 H100 GPU-hours for meta-training.*

1 INTRODUCTION

The remarkable achievements of deep neural networks are closely tied to the evolution of optimization techniques [36, 35, 1]. While hand-designed optimizers (e.g., Adam, Muon, etc.) remain the dominant choice in practice, learned optimizers (LOs), which use a small neural network to estimate parameter updates, have demonstrated the potential to discover superior optimization rules through meta-learning [2, 18, 5, 23, 24, 37].

A long line of work has sought to improve learned optimization mainly through architectural innovations and carefully designed inductive biases. Architecturally, LOs have evolved from LSTM-based update rules [2] and hierarchical RNNs [42] to LSTM-MLP hybrids [19], per-parameter MLPs [20], LSTM hypernetworks [21], and, more recently, per-parameter MLPs equipped with Newton-Schulz iterations [24]. Complementary work has improved the inductive bias of LOs by adding nominal descent terms, weight decay, and output preconditioning [9], incorporating Hessian and local-entropy flatness-aware regularizers [43], and placing the LOs in the μP parameterization to improve generalization [37]. Despite steady progress in these directions, existing meta-training algorithms still struggle to scale efficiently to long-horizon optimization learning. Moreover, the resulting LOs often underperform comparable hand-designed optimizers.

In this work, we discuss why traditional meta-training pipelines waste computation. To address this inefficiency, we use a resume buffer to effectively recycle compute from redundant early-unroll learning to later regions where long-horizon failures happen. However, this also introduces unstable meta-training dynamics due to compounding errors in long-unroll optimization. For stabilization and generalization purposes, we additionally constrain the LO with progressive expert guidance, which anchors early parameter updates to a reliable expert optimizer through separate direction and magnitude supervision, and gradually relaxes this constraint as the LO improves. Our contributions can be summarized as follows:

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.

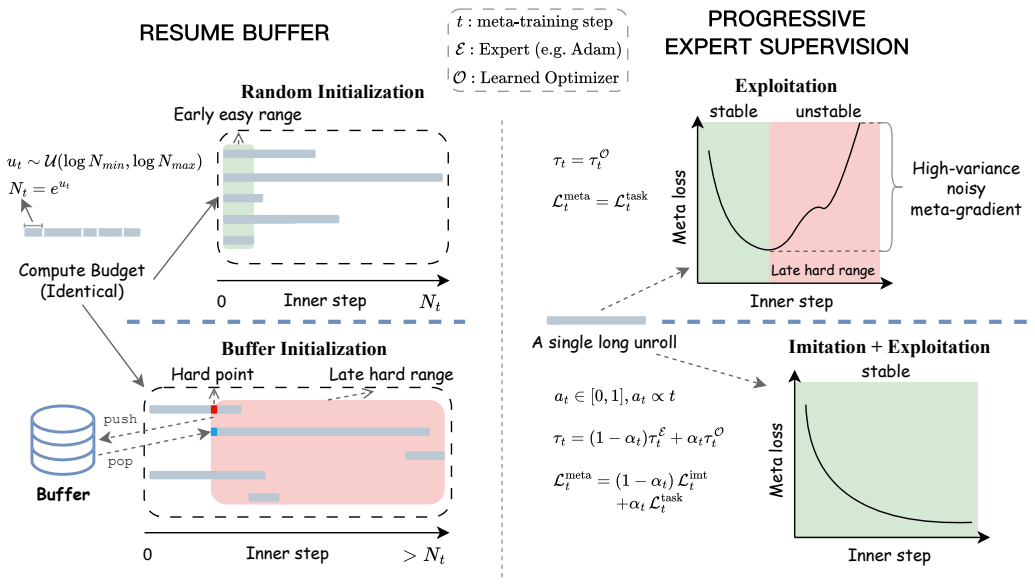

Figure 1: Overview of ELO, which combines a resume buffer with progressive expert supervision. **Left**: Under the same compute budget, the resume buffer restarts from stored states rather than random initialization, redirecting computation from easily learned early stages to later, failure-prone regions. **Right**: Progressive expert supervision begins with pure offline imitation learning, then alternates between imitation and exploitation before switching entirely to exploitation, stabilizing meta-training throughout. While imitation learning potentially improves the LO's task-agnostic generalization, the final exploitation stage encourages it to learn update rules that outperform the expert.

- We propose efficient long-horizon learning (ELO), a novel meta-training algorithm for learned optimization, which incorporates a resume buffer and progressive expert guidance. We empirically demonstrate that ELO achieves faster meta-training convergence than prior approaches under the same compute budget.
- We establish the performance of LOs meta-trained with ELO on practical downstream image classification and language modeling tasks, where ELO baselines consistently improve over their corresponding base LOs. Most notably, ELO-Celo2 outperforms well-tuned AdamW across all evaluated tasks and remains competitive with Muon on language modeling.
- Through extensive ablation studies, we establish the contribution of ELO's key components, including the improvements from the buffer, the effect of expert optimizer choice, and the benefit of a decoupled imitation objective over a standard MSE loss.

2 Background

We review the necessary background and related work on learned optimization to establish the context for our method. For an extended review of related literature, we refer the reader to Appendix B.

Learned Optimizer Architecture. In this work, we adopt two LO architectures: the state-of-the-art Celo2 [24] and the standard per-parameter MLP baseline `small_fc_lopt` [20], which we refer to as `small_fc` for brevity. `small_fc` is a three-layer MLP with ReLU activations. It takes as input a feature vector (e.g., gradient, momentum, rms, etc.) for each parameter θ of the optimizee (the network currently being optimized) and outputs an update direction, d, and magnitude, m. That is, $f_\phi(\cdot) = [d, m]$, where f_ϕ is the LO and ϕ denotes its parameters. Concretely, the update is applied as follows:

$$\theta_{n+1} = \theta_n - \Delta\theta_n; \quad \Delta\theta_n = \lambda_1 d \times e^{\lambda_2 m}, \tag{1}$$

where λ_1 and λ_2 are hyperparameters usually fixed to 0.001. Building on the element-wise update rule of `small_fc`, Celo2 introduces tensor-level orthonormalization as a key improvement for non-1D parameters in the optimizee. Specifically, for any weight matrix $\boldsymbol{W}$ in the optimizee, the update is applied as:

$$\boldsymbol{W}_{n+1} = \boldsymbol{W}_n - \text{NewtonShulz5}(\Delta_n)/\|\Delta_n\|_{\text{rms}}. \tag{2}$$

Meta-training Objective. In general, learning the meta-parameters, ϕ, involves solving an optimization problem of the form:

$$\min_{\phi} \ \mathbb{E}_{\tau\sim\mathcal{D}}\Big[\sum_{n=1}^{N} \mathcal{L}_n^{\text{meta}}(\theta_n;\phi,\tau)\Big], \tag{3}$$

where $\mathcal{D}$ is a distribution of tasks, τ specifies a task (e.g., optimizee initialization, dataset, objective). The objective seeks to minimize the sum of per inner step losses over the training horizon N [37].

Unbiased Truncated Gradient Estimation. Estimating meta-gradients of long unrolled computation graphs with backpropagation is prone to noise due to exploding or vanishing gradients [18]. As such, the learned optimization community has turned towards zeroth-order estimators to estimate meta-gradients, including full ES [18] and the more efficient truncated estimator PES [41], which we adopt in this work. PES is an unbiased estimator of the Gaussian-smoothed objective [41]. It estimates meta-gradients over segments of R inner steps, where $R \ll N$ and N denotes the full inner-problem horizon. With antithetic samples, the PES estimator at truncation index k is given by:

$$\hat{\boldsymbol{g}}_k^{\text{PES}} = \frac{1}{2\sigma^2 P}\sum_{i=1}^{P}\sum_{n=1}^{R} \xi_k^{(i)} \Delta\mathcal{L}_{n,i,k}^{\text{meta}}; \quad \Delta\mathcal{L}_{n,i,k}^{\text{meta}} = \mathcal{L}_n^{\text{meta}}(\phi+\epsilon^{(i,k)}) - \mathcal{L}_n^{\text{meta}}(\phi-\epsilon^{(i,k)}). \tag{4}$$

Where P is the number of monte-carlo estimates or particles, $\xi_k^{(i)} = \sum_{j=1}^{k}\epsilon^{(i,j)}$ is an accumulator of perturbations, and $\epsilon^{(i,j)}$ is the current perturbations sampled from $\mathcal{N}(0,\sigma^2)$. Since our resume buffer maintains the PES state, the meta-gradients resulting from resumed inner problems are also unbiased (See Appendix A for more details).

Sampling Inner-Loop Horizons. Existing work samples the inner unroll horizon mainly through log-uniform sampling [20, 37, 23, 24] or curriculum sampling [5]. In log-uniform sampling, each inner problem is assigned a horizon N sampled according to:

$$p(N) = \frac{1}{N\log(N_{\max}/N_{\min})}, \qquad N \in (N_{\min}, N_{\max}]. \tag{5}$$

In curriculum sampling, training starts from an initial horizon N_{init} and increases it over time:

$$N_j = \min\left(N_{\max}, N_{\text{init}} + \left\lfloor \frac{j}{N_{\text{period}}} \right\rfloor \Delta_N\right), \tag{6}$$

where j denotes the number of inner problem resets, N_{period} controls how often the horizon is increased, and Δ_N is the horizon increment. Larger values of $N_{\max}$, or more aggressive curriculum schedules, expose the LO to very long horizons since early meta-training, but also risk instability, similar to ELO-small_fc-B in **Figure 4** (Top). Therefore, in practice, relatively conservative settings are commonly used (**Figure 3** (Middle)).

3 ELO: Efficient Long-hOrizon learning

3.1 Inefficiency of Random Restarts

During meta-training, when the LO makes an update error ϵ_n at inner step n, then the state reached at a later step $m > n$ depends on the accumulated effect of all previous errors:

$$\theta_m - \theta_m^{\star} \approx \sum_{n=0}^{m-1} C_{n,m}\epsilon_n, \tag{7}$$

Algorithm 1: Efficient Long-hOrizon Learning (ELO)

for $t = 0, 1, \ldots, T-1$ **do**
  Sample $u \sim \text{Uniform}(0, 1)$;
  if $(u < P_{\mathcal{B}}) \wedge (\mathcal{B} \neq \emptyset)$ **then**
    $(K, s_K) \sim \text{Uniform}(\mathcal{B})$;
  else
    $K = 0,\ s_0 \leftarrow$ randomly initialized inner state;
  end
  $V_K \leftarrow 0$; $M_K \leftarrow 0$; $n^\star \leftarrow K$;
  $\alpha_t = \frac{t}{T-1}$;
  for $n = K, K+1, \ldots, K+N-1$ **do**
    $\theta^{\mathcal{E}}_{n+1} \leftarrow \theta_n - \Delta\theta^{\mathcal{E}}_n$; $\theta^{\mathcal{O}}_{n+1} \leftarrow \theta_n - \Delta\theta^{\mathcal{O}}_n$;
    $\theta_{n+1} \leftarrow (1-\alpha_t)\theta^{\mathcal{E}}_{n+1} + \alpha_t \theta^{\mathcal{O}}_{n+1}$;
    $\mathcal{L}^{\text{dir}}_n \leftarrow 1 - \frac{\Delta\theta^{\mathcal{E}}_n \cdot \Delta\theta^{\mathcal{O}}_n}{\|\Delta\theta^{\mathcal{E}}_n\|_2 \|\Delta\theta^{\mathcal{O}}_n\|_2}$;
    $\mathcal{L}^{\text{mag}}_n \leftarrow \left| \|\Delta\theta^{\mathcal{E}}_n\|_2 - \|\Delta\theta^{\mathcal{O}}_n\|_2 \right|$;
    $\mathcal{L}^{\text{imt}}_n \leftarrow \lambda \mathcal{L}^{\text{dir}}_n + (1-\lambda)\mathcal{L}^{\text{mag}}_n$;
    $\mathcal{L}^{\text{meta}}_n \leftarrow (1-\alpha_t)\mathcal{L}^{\text{imt}}_n + \alpha_t \mathcal{L}^{\text{task}}_n$;
    $\Delta^\ell_n \leftarrow \mathcal{L}^{\text{task}}_{n-1} - \mathcal{L}^{\text{task}}_n$ if $n > K$ else 0;
    $V_{n+1} \leftarrow V_n + \Delta^\ell_n$;
  end
  $n^\star \leftarrow \arg\max V_n$;
  $\mathcal{B} \leftarrow \text{Push}(\mathcal{B}, (n^\star, s_{n^\star}))$;
  $g_t \leftarrow \text{PES}\left(\sum_{n=K}^{K+N-1} \mathcal{L}^{\text{meta}}_n\right)$;
  $\phi_{t+1} \leftarrow \text{Update}(\phi_t, g_t)$;
end

Hyperparams:
Resume probability $P_{\mathcal{B}}$, buffer capacity C, direction loss weight λ.

Initialize:
Resume buffer $\mathcal{B} \leftarrow \emptyset$ with capacity C.

Notation:
ELO combines Resume Buffer and Progressive Expert Supervision.

T	meta-train steps
N	unroll length
K	inner start step
$\mathcal{B}$	resume buffer
s_n	inner state at step n
$\mathcal{E}$	expert optimizer
$\mathcal{O}$	learned optimizer (LO)
θ	optimizee parameters
ϕ	LO parameters
α_t	progressive schedule
V_n	progress score

where $\theta^\star_m$ represents the ideal optimal state at step m, and $C_{n,m}$ describes how an error made at step n propagates to the state at step m.

With less accumulated error, the inner task loss (e.g., CE Loss for image classification) decreases more steadily early in an unroll, while later steps are more likely to diverge. This is especially common early in meta-training, when the LO has not yet learned a reliable update policy. Such divergence can produce large meta-gradients that distort the meta-optimizer's running statistics, such as momentum. Even if the loss recovers, these effects can persist and interfere with subsequent meta-updates. We therefore treat these gradients as high-variance noise. In contrast, the cleaner, more stable meta-gradients from early steps generally allow faster learning of early-stage behavior.

Existing horizon sampling strategies, such as those described in **Sec. 2**, always restart each inner problem from random initialization, which biases meta-training toward early unroll steps. As a result, once early-unroll behavior has been well learned, random restarts continue to allocate a disproportionate fraction of the meta-training budget to these already well-covered regions, leaving later, failure-prone regimes under-trained.

3.2 Resume Buffer

Throughout meta-training, we maintain a resume buffer that lets future unrolls restart from challenging points in previous trajectories. Specifically, we identify these regions from the loss dynamics of the inner trajectory. Given inner losses $\ell_0, \ell_1, \ldots, \ell_N$, define the one-step loss improvement $\Delta^\ell_n = \ell_{n-1} - \ell_n$. We accumulate these local improvements into a running progress score, $V_n = V_{n-1} + \Delta^\ell_n$, with $V_0 = 0$, so that $V_n = \ell_0 - \ell_n$ measures the total progress made up to step n. After a full unroll, we select the step with the highest progress score, $n^\star = \arg\max_{0 \le n \le N} V_n$.

This step corresponds to the lowest loss reached during the unroll, after which the loss diverges or saturates. We then push the corresponding inner state $s_{n^\star}$ into the buffer. At the beginning of each new unroll, we randomly resume from one of the buffered states with probability $P_{\mathcal{B}}$ and otherwise start from a fresh random initialization. This shifts a controlled fraction of meta-training compute from already well-covered early stages to later regimes where long-horizon failures emerge. We set the buffer size to one by default, both for simplicity and to avoid stale states.

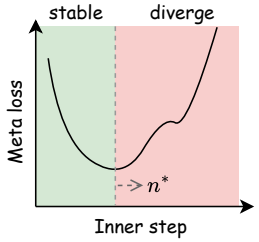

Figure 2: Push inner state to buffer at step n^*.

3.3 Progressive Expert Supervision

When applying aggressive resume buffer schedule (large $P_{\mathcal{B}}$), we observe unstable meta-training dynamics (ELO-small_fc-B in **Figure 4** Top). As discussed in **Sec. 2**, a large resume probability $P_{\mathcal{B}}$ can expose the LO to these failure-prone regions early in meta-training. The resulting noisy meta-gradients (**Figure 4** (Top)) can destabilize meta-training and, in severe cases, lead to collapse (**Figure 4 (Top)**).

To stabilize meta-training under a large $P_{\mathcal{B}}$ and encourage task-agnostic generalization learning, we regularize the LO with a reliable expert optimizer during early meta-training and progressively relax this constraint as the LO improves.

Trajectory fusion. For any inner step n, we fuse the trajectory produced by the expert and the one produced by the LO in the following way:

$$\begin{gathered}\theta^{\mathcal{E}}_{n+1} = \theta_n - \Delta\theta^{\mathcal{E}}_n; \qquad \theta^{\mathcal{O}}_{n+1} = \theta_n - \Delta\theta^{\mathcal{O}}_n; \\ \theta_{n+1} = (1-\alpha_t)\,\theta^{\mathcal{E}}_{n+1} + \alpha_t\,\theta^{\mathcal{O}}_{n+1},\end{gathered} \tag{8}$$

where $\mathcal{E}$ and $\mathcal{O}$ denote the expert and the LO, respectively. As meta-training progresses, the LO's updates become increasingly reliable; the schedule α_t correspondingly shifts weight from the expert to the LO, keeping the inner trajectory high-quality throughout meta-training.

Progressive Expert Supervision. When applying mixed trajectories, meta-gradients at the beginning of meta-training become uninformative, as they are estimated from pure expert trajectories ($\alpha_t = 0$), rather than trajectories induced by the LO itself. To address this, we leverage expert supervision to provide clean gradient signals for early meta-training. A natural choice is to imitate expert updates with a standard MSE loss, which couples errors in update direction and magnitude. We instead decouple direction and magnitude supervision and combine them with explicit weights for finer control over their relative importance, yielding the following learning objective:

$$\begin{gathered}\mathcal{L}^{\text{dir}}_n = 1 - \frac{\Delta\theta^{\mathcal{E}}_n \cdot \Delta\theta^{\mathcal{O}}_n}{\|\Delta\theta^{\mathcal{E}}_n\|_2 \, \|\Delta\theta^{\mathcal{O}}_n\|_2}; \qquad \mathcal{L}^{\text{mag}}_n = \left|\|\Delta\theta^{\mathcal{E}}_n\|_2 - \|\Delta\theta^{\mathcal{O}}_n\|_2\right|; \\ \mathcal{L}^{\text{imt}}_n = \lambda\mathcal{L}^{\text{dir}}_n + (1-\lambda)\mathcal{L}^{\text{mag}}_n.\end{gathered} \tag{9}$$

We define the meta-loss as $\mathcal{L}^{\text{meta}}_n = (1-\alpha_t)\,\mathcal{L}^{\text{imt}}_n + \alpha_t\,\mathcal{L}^{\text{task}}_n$, where α_t follows the same schedule as in **Eq.** 8. When $\alpha_t = 0$, the objective reduces to pure expert supervision. As α_t increases, the objective continuously interpolates between imitation and task optimization. At $\alpha_t = 1$, the LO is trained only through task-driven loss, allowing it to improve beyond the expert policy.

4 Empirical Evaluation

We describe the main experimental settings for evaluating ELO, with complete details provided in Appendix D.

Baselines. We adopt two representative MLP-based LOs, `small_fc` [20] and Celo2 [24]. For each architecture, we compare the ELO-trained variant, ELO-`small_fc` and ELO-Celo2, with its non-ELO counterpart. We also compare against CL-`small_fc`, a curriculum baseline with alternating offline imitation learning inspired by [5], and strong hand-designed optimizers including AdamW [17] and Muon [13].

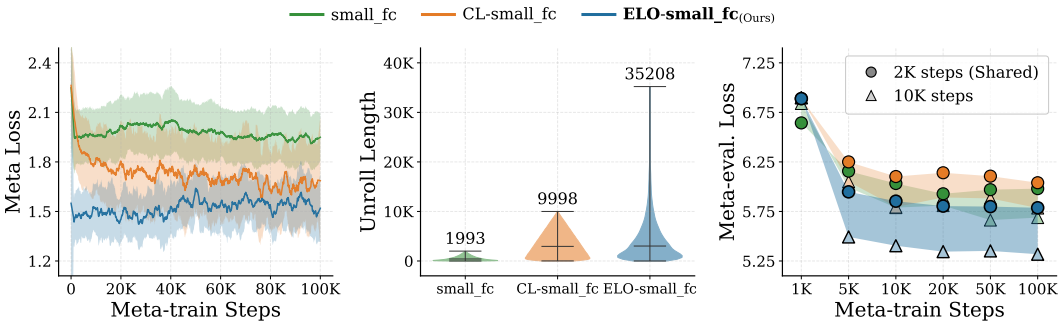

Figure 3: The meta-loss at each outer step is the average task loss over a full unroll or truncation window. ELO maintains a lower meta-loss by enabling learning over longer unrolls under a fixed compute budget, and achieves faster meta-training convergence at both the shared 2K-step horizon and the longer 10K-step horizon.

Meta-training. Our experimental setup and meta-training pipeline largely follow [20, 37, 23] with tuned hyperparameters applied for each baseline. All LOs are meta-trained on four simple 8×8 image classification tasks: MNIST, Fashion-MNIST, CIFAR-10, and SVHN. The optimizee is a one-layer MLP with width 32. We meta-train for 100K outer steps with batch size 64, using AdamW as the meta-optimizer with a cosine learning rate schedule. We search the outer learning rate over $\{10^{-3}, 3 \times 10^{-4}, 10^{-4}\}$ and use a weight decay of 10^{-4}. We estimate meta-gradients with Persistent Evolution Strategies (PES), using a truncation length of $R = 50$. We also apply task augmentation with range of [0.001, 1000] during meta-training, following [23].

Downstream Evaluation. We evaluate ELO on various practical vision and language model pre-training tasks. For vision experiments, We train on ResNet-50 [10] and ViT-Base/16 [7] on ImageNet-1K. For language modeling, we train GPT-2 (30M and 124M) and GPT-3 (1.3B) on FineWeb [27], using a budget of 20 tokens per parameter for the GPT-2 models [12] and 10B tokens for GPT-3. All methods use cosine learning rate schedules, and we carefully tune the learning rate and weight decay for each optimizer family.

4.1 Generalization to Practical Pre-training Tasks

Takeaway : ELO-LOs meta-trained at a small scale are able to generalize to models with hundreds of millions of parameters, consistently improving over their corresponding base LOs across multiple benchmarks. Most notably, ELO-Celo2 outperforms well-tuned AdamW across all tasks and remains competitive with Muon on language modeling at different scales.

All ELO baselines are meta-trained on four simple vision tasks using a one-layer MLP with width 32. We evaluate whether optimizers learned in this small meta-training setting can transfer to realistic optimization problems. We compare ELO-trained LOs with other LO baselines and hand-designed optimizers on various significantly larger scale vision and language model pre-training tasks.

Image classification. We train ViT-B/16 and ResNet-50 models on ImageNet-1K at 224×224 resolution for 50K steps with batch size 2048. We report top-1 validation accuracy in **Table** 1 on three validation benchmarks: ImageNet-1K validation, ImageNet-ReaL, and ImageNet-V2. LOs meta-trained with ELO achieve stronger performance than all LO baselines using the same architecture and outperform a strong AdamW baseline. As shown in **Figure** 6, we find that LOs require substantially stronger data augmentation than AdamW. Under the augmentation setting well tuned for AdamW, LOs tend to overfit, while increasing augmentation strength improves their validation performance. In contrast, AdamW naturally generalizes better under a lighter augmentation, and its performance degrades under stronger augmentation.

Language model pretraining. We evaluate ELO on out-of-distribution language modeling tasks on FineWeb (**Figure** 5). On GPT-2 (30M and 124M), trained for 0.6B and 2.5B tokens, respectively, both ELO-`small_fc` and ELO-Celo2 substantially improve over their corresponding base LOs. ELO-Celo2 achieves the lowest validation loss at both scales, even outperforming Muon. We further

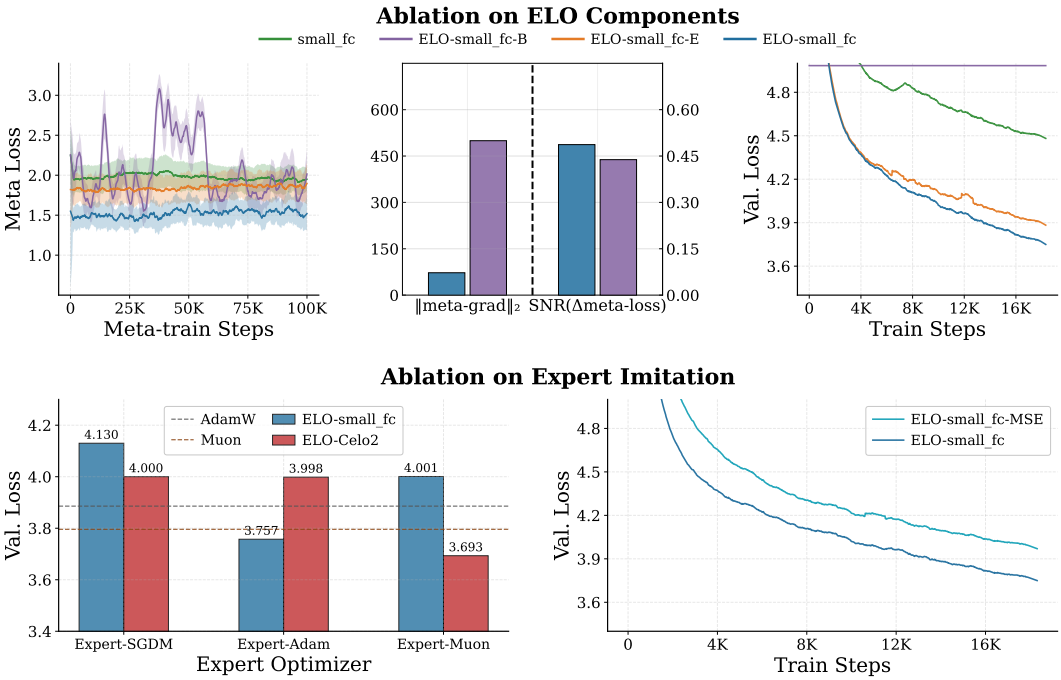

Figure 4: Ablations on meta-training task (four tiny image classification tasks using a one-layer MLP) and downstream GPT-2 (30M) pretraining for 0.6B tokens. **Top**: An aggressive resume buffer schedule leads to larger meta-gradients with lower signal-to-noise ratio. Expert supervision stabilizes training, while the resume buffer further improves downstream long-unroll performance. **Bottom**: For expert optimizer, AdamW works best for the element-wise `small_fc`, while Muon works best for the matrix-based Celo2. Separately weighting direction and magnitude supervision also improves downstream performance over standard MSE imitation.

evaluate ELO-Celo2 on GPT-3 (1.3B), trained for approximately 10B tokens, comparing it with strong learned and hand-designed baselines, including Celo2, AdamW, and Muon. While Muon achieves the lowest validation loss, ELO-Celo2 shows substantially better generalization than the base Celo2.

4.2 ELO Design Analysis

Takeaway : Resume buffer improves long-horizon coverage, while expert supervision keeps these harder unrolls stable. The optimal expert is architecture dependent, with Adam preferred for `small_fc` and Muon preferred for Celo2. Decoupling direction and magnitude supervision further outperforms direct ℓ_2 regression.

In this section, we conduct ablation and comparison experiments to understand how each design of ELO contributes and how they interact.

Ablating ELO's main components. We ablate the resume buffer and progressive expert supervision in **Figure** 4 (Top). Using only the resume buffer with an aggressive schedule (ELO-`small_fc`-B) destabilizes meta-training and leads to poor generalization to downstream task. The buffer exposes the LO to long-horizon inner problems before it has learned reliable updates, allowing errors to accumulate along the trajectory. Compared with full ELO, this variant produces much larger meta-gradient norms and a lower signal-to-noise ratio in the meta-loss differences (closely related to the quality of meta-gradient estimates) used by PES. In contrast, progressive expert supervision alone (ELO-`small_fc`-E) stabilizes meta-training and substantially improves downstream performance. Combining both components (ELO-`small_fc`) preserves this stability and further improves downstream optimization, with a growing advantage over expert supervision alone at later training steps.

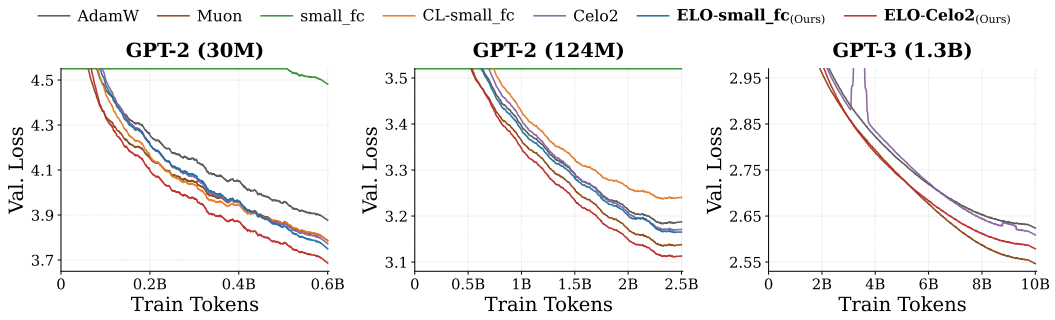

Figure 5: Validation loss of downstream GPT model pretraining on FineWeb. We compare all methods on GPT-2 (30M and 124M), while the GPT-3 (1.3B) comparison focuses on four strong optimizers: AdamW, Muon, Celo2, and ELO-Celo2. In all cases, ELO-trained optimizers substantially outperform their corresponding base LOs (`small_fc` and Celo2).

Expert optimizer selection. The choice of expert optimizer is important because the expert defines the update target used during imitation and is the only source of learning signal early on during meta-training. We search over several hand-designed optimizers independently for different LO architectures and compare the downstream performance of resulting optimizers on language model pre-training with GPT-2 (30M), trained for 0.6B tokens. As shown in **Figure** 4 (Bottom), Adam gives the best validation loss when used as the expert for the element-wise `small_fc`. In contrast, Muon works best as the expert for the matrix-based Celo2 optimizer. A likely explanation is that `small_fc` applies element-wise update rules and does not explicitly model matrix operations, making it difficult to imitate Muon's orthogonalization based update. Celo2, by contrast, contains matrix-based operations and can more naturally imitate Muon-like behavior.

Imitation objective. We compare our decoupled imitation objective with direct MSE regression between the updates produced by the LO and the expert optimizer. While MSE couples update direction and magnitude, our approach allows finer control over their relative weights. This leads to lower validation loss in downstream task (GPT-2 (30M) pretraining in **Figure** 4 (Bottom)).

4.3 ELO Hyperparameter Sensitivity

Takeaway : Imitating expert's update direction is more important than matching its magnitude. A large buffer probability is preferred in our meta-training settings.

ELO introduces two critical hyperparameters: 1) the buffer resume probability $P_{\mathcal{B}}$, which controls how often unrolls are initialized from buffered states, and 2) the direction loss weight λ, which balances direction and magnitude supervision. We tune both hyperparameters mainly using ELO-`small_fc`.

Direction vs. magnitude. We set $P_{\mathcal{B}} = 0$ to isolate the imitation objective from the buffer mechanism. We sweep the direction loss weight λ over $\{0.1, 0.2, \ldots, 0.9\}$ and select it according to meta-evaluation loss. As shown in Appendix 7, assigning weight 0.7 to the direction term and 0.3 to the magnitude term gives the best performance. This suggests that matching the expert update direction is more important than exactly matching its scale, although magnitude supervision remains useful for controlling the update size.

Buffer probability. $P_{\mathcal{B}}$ controls how aggressively meta-training compute is reallocated toward longer and harder regimes, which is critical for downstream long-horizon tasks. Larger values naturally increase the probability of training from later inner states, but can also make meta-training less stable. We search $P_{\mathcal{B}}$ over $\{0.1, \ldots, 0.9\}$. To reduce the sweep cost while selecting a value that generalizes to downstream tasks, we conduct a two-stage selection. We first select three candidates, $P_{\mathcal{B}} \in \{0.1, 0.5, 0.8\}$, based on meta-evaluation loss. We then select the final $P_{\mathcal{B}} = 0.8$ according to the validation loss of GPT-2 (124M, 2.5B tokens) pretraining in Appendix 8. In practice, smaller $P_{\mathcal{B}}$ is safer at small meta-training scale, while larger $P_{\mathcal{B}}$ becomes beneficial as meta-training becomes more stable and can support longer continuations.

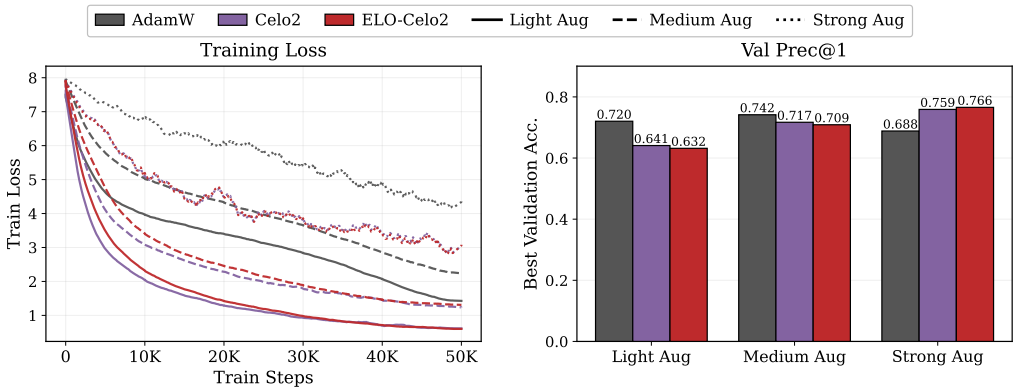

Figure 6: On vision tasks (e.g. ViT-B/16 on ImageNet-1K), learned optimizers achieve their best validation accuracy under stronger augmentation, using 6 randomly selected operations from a pool of more than 20, whereas hand-designed optimizers perform best under medium augmentation, using 4 randomly selected operations.

Table 1: Best validation accuracy across benchmarks. Top-1 accuracy on ImageNet-1K validation, ImageNet-ReaL, and ImageNet-V2 for ResNet-50 and ViT-B/16 trained from scratch, mean ± std over 3 seeds. The best mean in each column is in **bold**.

	ResNet-50			ViT-B/16			
Method	IN-1K	IN-Real	IN-V2	IN-1K	IN-Real	IN-V2	Avg.
AdamW	75.81±0.07	82.17±0.02	63.39±0.15	74.03±0.11	80.72±0.18	61.30±0.27	72.90
small_fc	65.31±0.10	73.66±0.03	53.51±0.11	32.10±0.94	37.09±1.10	25.69±0.88	47.89
CL-small_fc	71.91±0.30	79.23±0.26	59.58±0.30	67.65±0.22	74.88±0.16	54.73±0.27	68.00
Celo2	75.76±0.11	82.34±0.07	63.39±0.27	75.86±0.05	81.74±0.05	62.76±0.09	73.64
ELO-small_fc$_{\text{(Ours)}}$	74.08±0.16	81.13±0.12	62.10±0.30	75.15±0.07	81.69±0.05	62.08±0.10	72.70
ELO-Celo2$_{\text{(Ours)}}$	**76.11**±0.09	**82.70**±0.02	**64.03**±0.39	**76.42**±0.28	**82.26**±0.31	**63.89**±0.32	**74.24**

5 Discussion & Limitation

Due to computational constraints, we have not explored large-scale meta-training, which we believe is crucial for broader LO generalization. Our LOs are meta-trained on only four tiny image classification tasks, and we observe that they generalize less well across model scales than strong hand-designed optimizers. However, we believe that combining ELO with parameterizations such as μP [37] or CompleteP [6] has big potential to improve generalization across model scales under limited meta-training budgets. On the implementation side, ELO can introduce additional overhead compared with standard meta-training recipes (Appendix 2), mainly due to additional data movement across nested functions in JAX implementation. This overhead is more visible on tiny meta-training tasks, where GPU computation is small and CPU and memory operations become the main bottleneck. As the task scale increases, however, runtime becomes dominated by GPU computation, and ELO's relative overhead is expected to decrease substantially.

6 Conclusion

We proposed Efficient Long-hOrizon learning (ELO), a novel meta-training algorithm for learned optimization that combines progressive expert guidance with a resume buffer to achieve efficient long-horizon learning with stability and improve downstream performance. Across various vision and language-model benchmarks, ELO-small_fc and ELO-Celo2 outperform their corresponding learned-optimizer baselines, with ELO-Celo2 also outperforming AdamW and Muon on GPT2 (30M and 124M) pre-training. These results suggest that ELO is a promising recipe for scaling learned optimization.

AI USE STATEMENT

We used generative AI to help polish the writing, debug code, and prepare figures. These uses fall under the recommended disclosure categories in the ICLR 2027 AI Policy for Authors. We did not use generative AI for any tasks requiring disclosure. The research hypotheses, ELO design, experimental design, method implementations, and interpretation of results are our own work. This work did not involve synthetic data generation, dataset cleaning, translation, or qualitative data analysis. We take responsibility for the entire paper and its accompanying artifacts, including any AI-assisted content.

ETHICS STATEMENT

We study meta-training algorithms for learned optimizers. We do not involve human subjects in our experiments, nor do we collect or release new datasets. We use publicly available datasets, including MNIST, Fashion-MNIST, CIFAR-10, SVHN, ImageNet-1K, ImageNet-ReaL, ImageNet-V2, and FineWeb, in accordance with their respective licenses and usage conditions. We evaluate optimization performance on standard image classification and language modeling tasks. We recognize that publicly available datasets may still raise concerns about content, provenance, and bias, and we do not claim that our experiments resolve these concerns. We also recognize that the broader impact of our method depends on the models and applications in which it is used. We therefore encourage researchers and practitioners adopting our method to consider their data sources, intended uses, and potential effects on individuals and communities.

REPRODUCIBILITY STATEMENT

We describe the ELO meta-training procedure in Algorithm 1 and its main components in Secs. 3.2–3.3. Appendix A provides the assumptions and analysis for the unbiasedness of PES meta-gradients when resuming inner problems from the buffer. Experimental details, including task and model configurations, training budgets, and hyperparameter searches, are given in Sec. 4 and Appendix D. We report the selection of $P_{\mathcal{B}}$ and λ in Sec. 4.3 and Figures 7–8, and meta-training costs in Table 2. All datasets used in this work are publicly available. We will share an anonymized implementation with reviewers during the discussion period and release the code upon publication.

REFERENCES

[1] Ruslan Abdulkadirov, Pavel Lyakhov, and Nikolay Nagornov. Survey of optimization algorithms in modern neural networks. *Mathematics*, 11(11):2466, 2023.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[3] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

[4] Arslan Chaudhry, Albert Gordo, Puneet Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 6993–7001, 2021.

[5] Tianlong Chen, Weiyi Zhang, Jingyang Zhou, Shiyu Chang, Sijia Liu, Lisa Amini, and Zhangyang Wang. Training stronger baselines for learning to optimize. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[6] Nolan Dey, Bin 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. *Advances in Neural Information Processing Systems*, 38:137707–137739, 2026.

[7] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[8] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on robot learning*, pp. 158–168. PMLR, 2022.

[9] James Harrison, Luke Metz, and Jascha Sohl-Dickstein. A closer look at learned optimization: Stability, robustness, and inductive biases. *Advances in Neural Information Processing Systems*, 35:3758–3773, 2022.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[11] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3223–3230, 2018.

[12] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

[13] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.

[14] Oscar Li, James Harrison, Jascha Sohl-Dickstein, Virginia Smith, and Luke Metz. Variance-reduced gradient estimation via noise-reuse in online evolution strategies. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[15] Minghuan Liu, Hanye Zhao, Zhengyu Yang, Jian Shen, Weinan Zhang, Li Zhao, and Tie-Yan Liu. Curriculum offline imitating learning. *Advances in Neural Information Processing Systems*, 34:6266–6277, 2021.

[16] Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. In *2018 56th annual allerton conference on communication, control, and computing (Allerton)*, pp. 478–485. IEEE, 2018.

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

[18] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pp. 4556–4565. PMLR, 2019.

[19] Luke Metz, Niru Maheswaranathan, C. Daniel Freeman, Ben Poole, and Jascha Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves, 2020. URL https://arxiv.org/abs/2009.11243.

[20] Luke Metz, C Daniel Freeman, James Harrison, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Practical tradeoffs between memory, compute, and performance in learned optimizers. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2022.

[21] Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[23] Abhinav Moudgil, Boris Knyazev, Guillaume Lajoie, and Eugene Belilovsky. Celo: Training versatile learned optimizers on a compute diet. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=SLqJbt4emY.

[24] Abhinav Moudgil, Boris Knyazev, and Eugene Belilovsky. Celo2: Towards learned optimization free lunch. *arXiv preprint arXiv:2602.19142*, 2026.

[25] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, 2017.

[26] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179, 2018.

[27] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

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

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[30] Nived Rajaraman, Lin Yang, Jiantao Jiao, and Kannan Ramchandran. Toward the fundamental limits of imitation learning. *Advances in Neural Information Processing Systems*, 33:2914–2924, 2020.

[31] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019.

[32] 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.

[33] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[34] Simon Schmitt, Jonathan J. Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

[35] Ruo-Yu Sun. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China*, 8(2):249–294, 2020.

[36] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

[37] Benjamin Thérien, Charles Étienne Joseph, Boris Knyazev, Edouard Oyallon, Irina Rish, and Eugene Belilovsky. μlo: Compute-efficient meta-generalization of learned optimizers, 2024.

[38] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[39] Matej Vecerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017. This is often referred to as "DDPG from Demonstrations" (DDPGfD).

[40] Paul Vicol. Low-variance gradient estimation in unrolled computation graphs with es-single. In *International Conference on Machine Learning*, pp. 35084–35119. PMLR, 2023.

[41] Paul Vicol, Luke Metz, and Jascha Sohl-Dickstein. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10553–10563. PMLR, 2021.

[42] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International conference on machine learning*, pp. 3751–3760. PMLR, 2017.

[43] Junjie Yang, Tianlong Chen, Mingkang Zhu, Fengxiang He, Dacheng Tao, Yingbin Liang, and Zhangyang Wang. Learning to generalize provably in learning to optimize, 2023. URL https://arxiv.org/abs/2302.11085.

[44] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

A PES GRADIENTS REMAIN UNBIASED WHEN RESUMING FROM THE BUFFER

Following [42], throughout this section T denotes the unroll horizon and N the number of Monte Carlo samples. In the following sections, For the reader's convenience, we will now restate background from [41] required to understand our proof. Vicol et al. [41] derive *Persistent Evolution Strategies* (PES) for unrolled computation graphs whose total loss is $L(\boldsymbol{\theta}) = \sum_{t=1}^{T} L_t(\boldsymbol{\theta})$, where shared parameters $\boldsymbol{\theta} \in \mathbb{R}^P$ appear at every timestep. To account for the contribution of each application of $\boldsymbol{\theta}$, they introduce per-timestep copies $\boldsymbol{\theta}_t = \boldsymbol{\theta}$ and stack them as $\Theta = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T)^\top$; writing $L_t(\boldsymbol{s}_t; \boldsymbol{\theta})$ as $L_t(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t)$ (or simply $L_t(\Theta)$) implies $L_t(\cdot)$ is independent of $\boldsymbol{\theta}_\tau$ for $\tau > t$. Let $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}_1, \dots, \boldsymbol{\epsilon}_T)^\top$ be a matrix of perturbations with entries i.i.d. Gaussian of variance σ^2, let $\otimes$ denote the Kronecker product, and define $\boldsymbol{\xi}_t = \sum_{\tau=1}^{t} \boldsymbol{\epsilon}_\tau$. Starting from the full gradient $\frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)} \in \mathbb{R}^{PT \times 1}$ and using ES,

$$\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \sum_{\tau=1}^{T} \frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_\tau} = (\mathbf{I} \otimes \mathbf{1}^\top) \frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)}, \tag{10}$$

$$\boldsymbol{g}^{\text{PES}} = (\mathbf{I} \otimes \mathbf{1}^\top)\, \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{1}{\sigma^2} \operatorname{vec}(\boldsymbol{\epsilon})\; L(\Theta + \boldsymbol{\epsilon}) \right] \tag{11}$$

$$= \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}} \left[\Big(\sum_{\tau=1}^{T} \boldsymbol{\epsilon}_\tau \Big) L(\Theta + \boldsymbol{\epsilon}) \right]. \tag{12}$$

This ES approximation is an unbiased estimator of the gradient of the Gaussian-smoothed objective $\mathbb{E}_{\boldsymbol{\epsilon}}[L(\Theta + \boldsymbol{\epsilon})]$, and it decomposes sequentially as $\boldsymbol{g}^{\text{PES}} = \mathbb{E}_{\boldsymbol{\epsilon}} \left[\sum_{t=1}^{T} \hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}} \right]$ with $\hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}} = \frac{1}{\sigma^2} \boldsymbol{\xi}_t\, L_t(\boldsymbol{\theta}_1 + \boldsymbol{\epsilon}_1, \dots, \boldsymbol{\theta}_t + \boldsymbol{\epsilon}_t)$. The Monte Carlo PES estimator is then

$$\hat{\boldsymbol{g}}^{\text{PES}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}^{(i)}}.$$

In practice, Vicol et al. [41] use antithetic sampling,

$$\hat{\boldsymbol{g}}^{\text{PES-A}} \approx \frac{1}{2\sigma^2 N} \sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{\xi}_t^{(i)} \Big(L_t(\Theta + \boldsymbol{\epsilon}^{(i)}) - L_t(\Theta - \boldsymbol{\epsilon}^{(i)}) \Big).$$

Finally, they show PES is exactly unbiased for quadratic objectives:

Statement A.1 (PES is unbiased restated from [41])**.** *Let $\boldsymbol{\theta} \in \mathbb{R}^P$ and $L(\boldsymbol{\theta}) = \sum_{t=1}^{T} L_t(\boldsymbol{\theta})$. Suppose that $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ exists, and assume that L is quadratic, so that it is equivalent to its second-order Taylor series expansion: $L(\Theta + \boldsymbol{\epsilon}) = L(\Theta) + \operatorname{vec}(\boldsymbol{\epsilon})^\top \nabla_{\operatorname{vec}(\Theta)} L(\Theta) + \frac{1}{2} \operatorname{vec}(\boldsymbol{\epsilon})^\top \nabla^2_{\operatorname{vec}(\Theta)} L(\Theta) \operatorname{vec}(\boldsymbol{\epsilon})$. Then,* $\operatorname{bias}(\hat{\boldsymbol{g}}^{\text{PES-A}}) = \mathbb{E}_{\boldsymbol{\epsilon}}[\hat{\boldsymbol{g}}^{\text{PES-A}}] - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \mathbf{0}$.

When meta-training with ELO, we use an identical gradient estimator $\hat{\boldsymbol{g}}^{\text{PES-A-Buffer}} \equiv \hat{\boldsymbol{g}}^{\text{PES-A}}$, with the only difference being that some particles may have been resumed from the particle replay buffer. This has the effect of distancing the subsequent $\boldsymbol{\theta}_i$. However, by the definition of Θ the resulting gradient is unaffected by this distancing.

Corollary A.2 (PES gradients from the replay buffer are unbiased)**.** *Mirroring the assumptions from Statement A.1 and additionally assuming that the full PES state is preserved within the buffer and subsequently resumed from, then* $\operatorname{bias}(\hat{\boldsymbol{g}}^{\text{PES-A-Buffer}}) = \mathbb{E}_{\boldsymbol{\epsilon}}[\hat{\boldsymbol{g}}^{\text{PES-A-Buffer}}] - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \mathbf{0}$.

Proof. Since Statement A.1 has no temporal constraint on Θ, it directly applies to our case where particles are restarted from the buffer. □

B EXTENDED BACKGROUND

We now review additional backgrounds and related works beyond those discussed in section 2.

Input Features for Learned Optimizers. Taking the small_fc_lopt for example, which applies the same MLP independently to every parameter, and it is shape-agnostic. For each parameter, it receives a 39-dimensional feature vector:

Feature	Dim	Definition
Gradient	1	g_n
Parameter value	1	p_n
Momentum	3	$m_{\beta,n} = \beta m_{\beta,n-1} + (1-\beta) g_n, \quad \beta \in \mathcal{B} = \{0.9, 0.99, 0.999\}$
Second moment	1	$v_{\gamma,n} = \gamma v_{\gamma,n-1} + (1-\gamma) g_n^2, \quad \gamma \in \mathcal{G} = \{0.95\}$
Momentum-preconditioned	3	$m_{\beta,n} \big(v_{\gamma,n} + 10^{-6}\big)^{-1/2}$
Inverse second moment	1	$\big(v_{\gamma,n} + 10^{-6}\big)^{-1/2}$
Normalized gradient	3	$g_n \big(R_{\delta,n} / \operatorname{mean}(R_{\delta,n})\big)^{-1/2} C_{\delta,n}^{-1/2}, \quad \delta \in \mathcal{D}$
Row second moment	3	$R_{\delta,n} = \delta R_{\delta,n-1} + (1-\delta) \operatorname{mean}_{\text{col}}(g_n^2), \quad \delta \in \mathcal{D}$
Column second moment	3	$C_{\delta,n} = \delta C_{\delta,n-1} + (1-\delta) \operatorname{mean}_{\text{row}}(g_n^2), \quad \delta \in \mathcal{D}$
Inverse row second moment	3	$\big(R_{\delta,n} + 10^{-8}\big)^{-1/2}, \quad \delta \in \mathcal{D}$
Inverse column second moment	3	$\big(C_{\delta,n} + 10^{-8}\big)^{-1/2}, \quad \delta \in \mathcal{D}$
Factored-preconditioned momentum	3	$m_{\beta,n} \big(R_{\delta,n} / \operatorname{mean}(R_{\delta,n})\big)^{-1/2} C_{\delta,n}^{-1/2}$
Step embedding	11	$\tanh(n/\tau - 1), \quad \tau \in \{1, 3, 10, 30, 100, 300, 10^3, 3000, 10^4, 30000, 10^5\}$
Total	39	—

The row/column statistics are computed at the tensor level and broadcast to each element, so small_fc still receives minor tensor-level information despite operating element-wise. The network architecture is $39 \to 32 \to 32 \to 2$ with ReLU activations. It outputs an update direction and magnitude. The same interface is used for all parameter types, including biases, embeddings, normalization parameters, convolution kernels, and non-matrix tensors.

Learned Optimization. Learned optimizers (LOs) aim to replace hand-crafted optimizers with small neural networks. In incipient work, Andrychowicz et al. [2] show that an LSTM can successfully be used to optimize small deep learning problems to a better loss than SGD or Adam. Follow-up research proposed new gradient estimators [18, 41, 40, 14], introduced new architectures and features [21, 20], proposed low cost meta-training recipes [23, 24, 37], and introduced stabilizing mechanisms to improve generalization [5, 9]. Of these works, Chen et al. [5] is the most closely related to our own as their proposed meta-training scheme also leverages behaviour cloning. However, unlike our work, they (1) behaviour clone using an alternating optimization scheme instead of adding the behavior cloning term to the meta-loss and (2) they do not use a replay buffer.

Replay Buffers. Replay buffers are widely used in reinforcement learning (RL) [32, 16, 44] and continual learning (CL) [31, 4]. They store trajectories collected during training, enabling the learner to sample from past experiences rather than relying solely on the most recent data. In RL, this helps break temporal correlations [22], improves sample efficiency [33, 34, 39, 11, 25], and stabilize learning, while in CL it plays a key role in alleviating catastrophic forgetting [31, 3]. Unlike conventional replay buffers for CL or RL, which typically contain existing training examples, we leverage a particle buffer that contains inner-problem checkpoints from previous unrolls, allowing existing particles to restart from the buffer instead of random initialization.

Behavior Cloning. Behavior Cloning (BC) [30, 38] is one of the most widely used paradigms in imitation learning [28, 26, 15], where the objective is to approximate an expert policy by directly regressing from observed states to expert actions. Formally, given a dataset of expert demonstrations $\mathcal{D} = \{(x_i, a_i^\star)\}_{i=1}^N$, where $x_i \in \mathcal{X}$ denotes the state and $a_i^\star \in \mathcal{A}$ is the corresponding expert action, the goal is to learn a policy $\pi_\theta : \mathcal{X} \to \mathcal{A}$, parameterized by θ, that closely approximates the expert's

behavior. A simple version of behavior cloning solves the supervised regression problem [8]:

$$\hat{\pi}_\theta = \arg\min_{\pi_\theta} \frac{1}{N} \sum_{i=1}^{N} \|\pi_\theta(x_i) - a_i^\star\|_2^2 . \tag{13}$$

In our work, we leverage progressive behavior cloning primarily as a stabilizer for buffer-based meta-training, enabling efficient long-unroll meta-training.

C Wall-clock Optimizer steps and meta-training time

Table 2: **Wall-clock training time of different learned optimization algorithms on a single H100 GPU.** Each algorithm is meta-trained for 100,000 outer steps following the recipes outlined in Sec. 4. We report the median per-step time, computed as the median of consecutive `_runtime` deltas logged on W&B; total time is computed as median $\times$ steps. Δ is the relative overhead over the same-family baseline, either `small_fc` or `Celo2`. Although these runs are measured on H100, our meta-training tasks are small and the runtime is mainly **limited by CPU operations and data movement**, so using more accessible GPUs such as L40S gives nearly unchanged wall-clock time. As the per-step optimizee computation increases, e.g., in the Celo2 setting, ELO's relative overhead becomes substantially smaller.

Method	LO arch.	Median (s/step)	Total wall-time	Δ
SMALL_FC	small_fc	0.116	3h14m	–
CL-SMALL_FC	small_fc	0.122	3h23m	+4.6%
ELO-SMALL_FC	small_fc	0.158	4h23m	+35.6%
CELO2	Celo2	0.187	5h11m	–
ELO-CELO2	Celo2	0.231	6h25m	+23.8%

D Extended Empirical Settings

This section complements sec. 4 from the main manuscript with additional experimental details.

Meta-training. For `small_fc_lopt` and Celo2, we sample the unroll length from $[100, 2000]$ using log-uniform sampling. For CL-`small_fc_lopt`, we use Adam as the expert optimizer. We evaluate maximum meta-training unroll lengths of 10K and 30K, and find that 10K is the most stable setting and gives the best downstream performance. For ELO variants, we sample the base unroll length uniformly from $[100, 2000]$ and use the proposed failure-aware resume buffer to resume from difficult regions of the current optimization trajectory. We first evaluate learned optimizer checkpoints on ImageNet-1K (32×32), using a three-layer MLP with width 128 as the optimizee. Each optimizer is evaluated for 10K inner steps with batch size 4096. This evaluation serves as an intermediate meta-evaluation benchmark before transferring the learned optimizers to larger downstream workloads.

Image classification. For realistic vision pre-training, we evaluate on ImageNet-1K at 224×224 resolution using ResNet-50 [10] and ViT-Base/16 [7]. We compare learned optimizers mainly against AdamW [17], which remains the dominant trend for vision tasks. Training runs for 50K steps with batch size 2048 under a cosine learning-rate schedule and weight decay searched from $\{0.1, 0.01, 0.001, 0.0001\}$. For AdamW, we sweep seven learning rates on a logarithmic grid in $[10^{-2}, 10^{-5}]$ for ViT-Base/16 and in $[10^{-1}, 10^{-4}]$ for ResNet-50. For all learned optimizers, we sweep seven learning rates logarithmically in $[10^{-2}, 10^{-5}]$. We also sweep the data augmentation strength for each method, ranging from light augmentation (e.g., horizontal flip, random crop) to stronger combinations (e.g., randomly select six augmentations from a pool of more than twenty operations).

Language model pre-training. For language modeling, we compare learned optimizers with AdamW and the matrix-based optimizer Muon [13]. We pretrain GPT-2 models [29] with 30M and 124M parameters, as well as GPT-3 (1.3B), on FineWeb [27]. The GPT-2 models use token budgets of 0.6B and 2.5B, respectively, corresponding to approximately 20 tokens per parameter [12]. The

30M model is trained for 18,311 steps with an effective batch size of 64 and a sequence length of 512, while the 124M model is trained for 19,074 steps with an effective batch size of 128 and a sequence length of 1024. For GPT-3 (1.3B), we use sequence length 2048 and an effective batch size of 256, training for 19,274 steps on approximately 10B tokens. All methods use a cosine learning rate schedule. For each method, we tune weight decay on GPT-2 (124M) pretraining, selecting from $\{0.1, 0.01, 0.001, 0.0001\}$. We sweep seven logarithmically spaced learning rates in $[10^{-5}, 10^{-2}]$ for learned optimizers and AdamW, and in $[10^{-4}, 10^{-1}]$ for Muon.

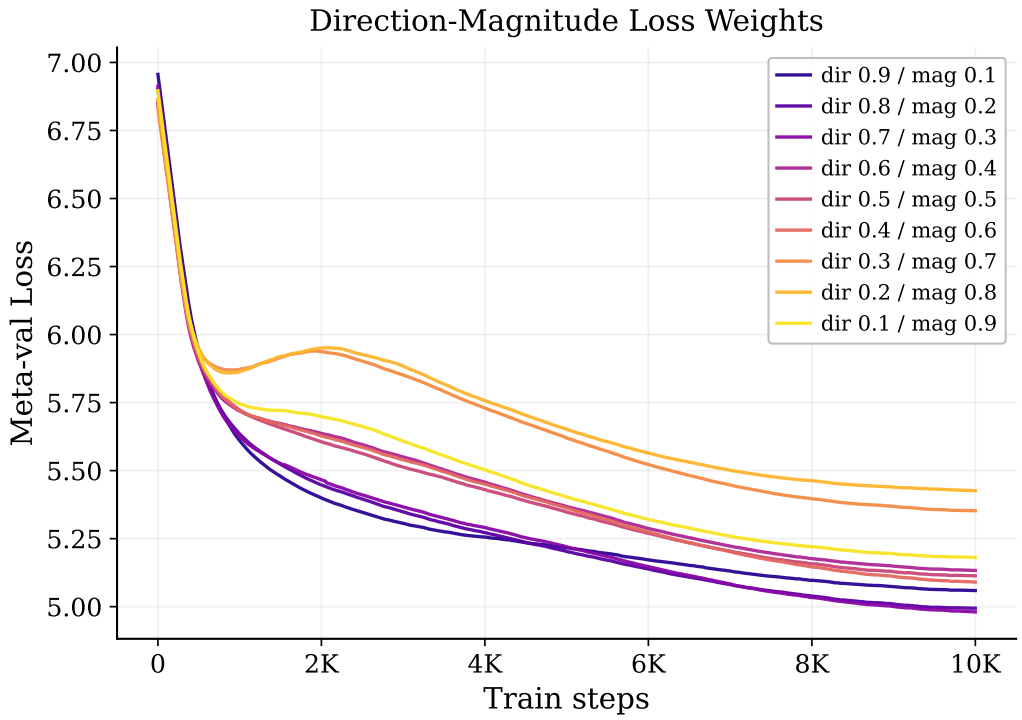

Figure 7: Effect of direction and magnitude weights in the imitation objective. We sweep the weighting between direction loss and magnitude loss and report meta-validation loss during training. Direction-dominant objectives consistently achieve lower meta-validation loss, while overly large magnitude weights slow convergence and lead to worse final performance. This supports decoupling update direction and scale rather than using a single regression loss on the full expert update.

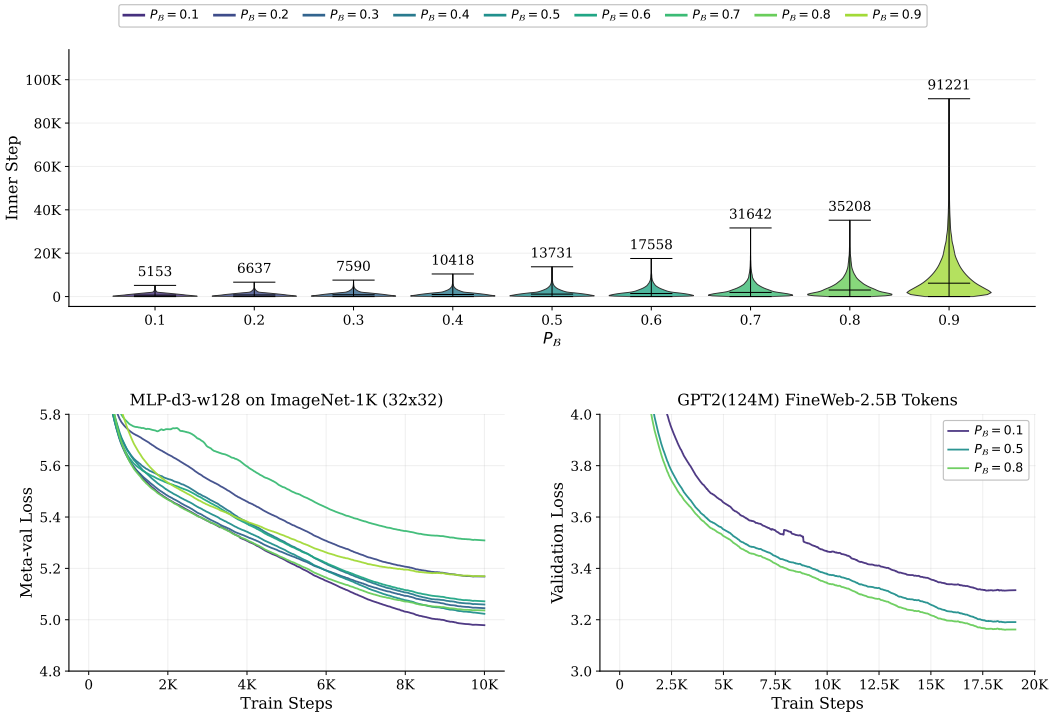

Figure 8: Effect of the resume probability $P_{\mathcal{B}}$ on long-horizon sampling and downstream performance. Top: increasing $P_{\mathcal{B}}$ shifts the sampled inner steps toward later regions of the trajectory, with the maximum reached step increasing from 5,153 at $P_{\mathcal{B}} = 0.1$ to 91,221 at $P_{\mathcal{B}} = 0.9$. Bottom left: on the ImageNet-1K 32×32 meta-evaluation task, smaller $P_{\mathcal{B}}$ generally gives better meta-validation loss, with the lowest loss achieved at $P_{\mathcal{B}} = 0.1$ Bottom right: on GPT-2 pre-training with 124M parameters on FineWeb for 2.5B tokens, larger $P_{\mathcal{B}}$ improves validation loss, indicating that practical long-horizon workloads benefit from more frequent resume-based sampling.