JUST ENOUGH LEARNING: GRPO-GUIDED CONTROLLERS FOR HYPERPARAMETER SWEEPS

Justin H. Lee
Cactus Compute Inc.
justin@cactuscompute.com

Henry Ndubuaku
Cactus Compute Inc.
henry@cactuscompute.com

ABSTRACT

Hyperparameter optimization remains a persistent bottleneck in deep learning, requiring expensive sweeps for each new model or dataset. We propose JEL (Just Enough Learning), a lightweight learned controller that adjusts optimizer hyperparameters throughout training. JEL treats the training process as an episodic reinforcement learning problem: at fixed decision intervals, a compact policy network observes training progress and outputs multiplicative corrections to learning rate and weight decay applied on top of a strong base optimizer. We train the controller using a modified group-relative policy optimization (GRPO) objective that removes length and per-group variance normalizations to avoid biasing the learning signal. On transformer pretraining tasks, JEL improves validation performance by 2.5% over schedule-free optimizers at equivalent computational cost, requiring controller training equivalent to only 5.6 training runs, a one-time cost amortized across deployments. JEL achieves performance within 8% of an upper bound from extensive manual experimentation, while already costing less than traditional 6-8 run hyperparameter sweeps, with savings compounding on each subsequent task. Our results demonstrate that a simple learned controller can effectively replace costly hyperparameter searches while maintaining competitive performance.

1 INTRODUCTION

Training modern deep networks requires careful hyperparameter tuning, particularly for learning rate schedules and weight decay. Despite advances in optimizer design, practitioners still rely on expensive grid searches, Asynchronous Successive Halving Algorithm (ASHA) sweeps (Li et al., 2020), or manual tuning, repeating this process for each new model architecture or dataset. This overhead grows with model size and computational budgets.

We propose JEL (Just Enough Learning), a learned hyperparameter controller that adapts optimizer settings during training without requiring per-task sweeps. JEL frames training as an episodic reinforcement learning problem: at fixed decision intervals, a compact policy network observes training progress and outputs multiplicative corrections to learning rate and weight decay, applied on top of a strong base optimizer such as Muon (Liu et al., 2025) or AdamW (Loshchilov & Hutter, 2019). The controller learns from experience across multiple training episodes, accumulating knowledge about effective optimization trajectories. The base optimizer provides robust default update rules, while the controller applies learned adjustments shaped by experience across many training runs.

1.1 MOTIVATION

Hyperparameter optimization fundamentally requires understanding training dynamics, yet most methods treat it as a black-box search problem detached from the optimization trajectory. Recent schedule-free optimizers (Defazio et al., 2024) eliminate explicit learning rate schedules but still require top-level configuration and offer no learned scheduling from past experience. Meanwhile, policy gradient methods with group-relative baselines (Shao et al., 2024) have shown promise for training small policies with stable learning signals.

JEL combines these insights: we use a lightweight learned controller trained via reinforcement learning to make hyperparameter decisions throughout training. The key challenges are (1) defining

rewards that encourage generalization rather than overfitting to the training set, (2) training the controller efficiently without introducing prohibitive meta-optimization costs, and (3) ensuring the learned policy transfers across different initialization seeds and training configurations. Unlike traditional per-task optimization, the controller distills experience from hundreds of training episodes into a reusable policy.

1.2 CONTRIBUTIONS

We propose JEL, a hyperparameter controller applying residual multiplicative adjustments to learning rate and weight decay, enabling learned adaptation while preserving base optimizer stability. We introduce a modified GRPO objective removing length and variance normalizations to prevent bias while maintaining training stability, and demonstrate that a simple 1D state representation (training progress only) suffices for effective control. JEL outperforms schedule-free optimizers by 2.5% in validation BPB with controller training costing only 5.6 runs, immediately competitive with traditional sweeps while reaching within 8% of hand-tuned upper bounds.

2 RELATED WORK

Hyperparameter optimization. Traditional approaches include grid search, random search, and Bayesian optimization. More recent methods like ASHA (Li et al., 2020) and BOHB (Falkner et al., 2018) use successive halving to efficiently allocate resources, while multi-fidelity Bayesian optimization (Do & Zhang, 2023) leverages cheaper approximations to guide search. However, these methods require substantial computational overhead, often training dozens or hundreds of configurations, and must be repeated for each new task.

Schedule-free and learned optimizers. Recent work eliminates explicit learning rate schedules (Defazio et al., 2024) but still requires careful base hyperparameter tuning. Meta-learning approaches train optimizers generalizing across tasks (Thérien et al., 2024) but require substantial meta-training compute and may struggle beyond their training distribution. JEL takes a complementary approach: learning lightweight controllers that adjust existing optimizers rather than replacing them.

RL for hyperparameter tuning. Xu et al. (2019) use RL to learn an adaptive learning rate schedule, with the controller observing rich training dynamics (loss, gradient statistics, weight distributions) to make closed-loop decisions for image classification. JEL instead discovers that a minimal state (training progress only) suffices when combined with multiplicative corrections on a strong base optimizer, while extending to joint learning rate and weight decay control for transformer pretraining.

Group-relative policy optimization. GRPO (Shao et al., 2024) introduces group-relative baselines for training policies with improved stability over vanilla PPO. We modify GRPO by removing length and variance normalizations that can bias learning, which proves essential for our hyperparameter controller.

3 METHOD

JEL treats training as a control problem where a learned policy makes hyperparameter adjustments throughout training. We formulate this as an episodic RL environment, train the controller with a modified GRPO objective, and deploy it alongside a base optimizer during actual training runs.

3.1 PROBLEM FORMULATION

We model training as a Markov Decision Process (Figure 1) where episodes are truncated training runs from random initialization to T steps, states s_t are compact representations of training progress, actions $a_t \in \mathbb{R}^2$ are continuous-valued adjustments to learning rate and weight decay, and rewards r_t reflect loss improvement on held-out data (a fixed probe set from the training split, separate from gradient updates). At each decision point (every $\Delta = 50$ steps), the controller observes the current state and outputs an action applied multiplicatively to the base optimizer's hyperparameters for the next Δ steps.

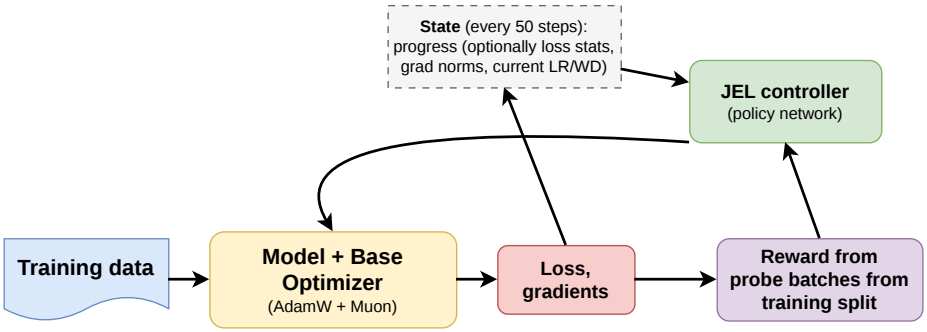

Figure 1: JEL system architecture. The controller observes training progress at regular intervals and applies multiplicative adjustments to the base optimizer's hyperparameters.

3.2 State Representation

After extensive experimentation, we found that a minimal 1D state suffices: $s_t = [t/T]$ where t is the current step and T is episode length, representing training progress in $[0, 1]$. We explored richer representations including gradient statistics, loss histories, and momentum buffers, but these consistently degraded performance, suggesting the reward signal already provides sufficient feedback about optimization quality.

3.3 Action Space and Mapping

The controller outputs raw actions $a_t = [\alpha_t, \beta_t] \in \mathbb{R}^2$ mapped to multiplicative scales: $\text{lr\_scale}_t = \exp(\lambda \cdot \tanh(\alpha_t))$ and $\text{wd\_scale}_t = \exp(\lambda \cdot \tanh(\beta_t))$ where $\lambda = \log(100) \approx 4.605$. The $\tanh$ bounds actions to $[-1, 1]$ and exponential mapping ensures positive scales in $[0.01, 100]$, making the controller scale-invariant. These multiply base hyperparameters: $\text{lr}_t = \text{lr}_{\text{base}}(t) \cdot \text{lr\_scale}_t$. The 100× range provided necessary expressiveness without instability; expanding to per-optimizer control degraded performance.

3.4 Reward Function

The reward at decision point t is $r_t = (L_{t-1} - L_t) - \lambda_g \cdot g_t - \lambda_s \cdot \|\tanh(a_t) - \tanh(a_{t-1})\|^2$, where L_t is average loss on a fixed probe set, g_t is gradient norm, and λ_g, λ_s are penalty weights. The primary term rewards loss reduction, directly optimizing for training quality. In our final experiments, we set $\lambda_g = \lambda_s = 0$ after finding that direct loss improvement alone provided sufficient signal. We compute L_t using a fixed probe set of 4 batches drawn from the training split via a separate iterator, ensuring the dedicated validation set is never used during meta-training and remains uncontaminated for final evaluation.

3.5 Modified Group-Relative Policy Optimization

We train the policy using PPO (Schulman et al., 2017) with group-relative advantages. Standard GRPO (Shao et al., 2024) includes length and variance normalizations that can bias learning when returns correlate with episode properties.

Our modification computes advantages as:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

$$b_k = \frac{1}{|G_k|} \sum_{i \in G_k} R_i^{(k)} \tag{2}$$

$$A_t = R_t - b_{k(t)} \tag{3}$$

where R_t is the empirical return from step t, G_k is the set of all samples at step-in-episode k across the rollout batch, and b_k is the group baseline for step k. Crucially, we do *not* normalize by episode length or standardize per-group, unlike standard GRPO.

The policy $\pi_\theta(a|s)$ is parameterized as a Gaussian with state-dependent mean and standard deviation output by a 2-layer MLP with 64 hidden units and tanh activations. We optimize the clipped PPO objective:

$$L_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{4}$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the importance sampling ratio and $\hat{A}_t = A_t$ are the group-relative advantages defined above (without further normalization). We use $\epsilon = 0.2$, batch size 64, and 4 PPO epochs per update.

We empirically found that removing length and variance normalizations maintained stable entropy throughout training, while standard GRPO exhibited entropy collapse after initial updates. This stability was critical for learning effective policies.

4 Experiments

We evaluate JEL on transformer pretraining tasks using the nanochat framework, comparing against hand-tuned baselines, schedule-free optimizers, and suboptimal configurations that demonstrate JEL's impact.

4.1 Experimental Setup

Model architecture. We train a 6-layer decoder-only transformer with dimension 384, 3 attention heads (head dimension 128), and 73.5M parameters.

Training configuration. All models train for 561 steps with 524K tokens/step (294M total) using hybrid AdamW+Muon: AdamW ($\beta_1 = 0.8, \beta_2 = 0.95$) for embeddings, Muon for matrices, with warmup-flat-warmdown scheduling (40% warmdown). The hand-tuned upper bound uses nanochat defaults: per-group LRs (embedding 0.3, Muon 0.02) and WD 0.2 scaled by $(12/\text{depth})^2$ to 0.8. JEL and the suboptimal baseline instead use a uniform conservative base (LR 10^{-3}, WD 10^{-2} for all parameter groups), deliberately avoiding per-group tuning so the controller can learn appropriate scaling via multiplicative adjustments.

Data. We use nanochat pretraining data (vocab 32,768), reporting validation bits-per-byte (BPB).

Baselines. We compare four configurations: (1) Schedule-Free Baseline from an 8-configuration sweep (LR 2.0, no WD, no warmup), at comparable cost to JEL's 5.6 run equivalents; (2) JEL with default meta-hyperparameters; (3) Hand-Tuned Upper Bound from extensive manual experimentation; (4) Suboptimal Baseline with controller disabled (constant 1× multiplier).

JEL controller training. The controller is trained for 50 updates with 8 episodes per update (distributed across GPUs), each running 561 steps at reduced sequence length (128 vs. 2048 tokens) with decisions every 50 steps. We use modified GRPO with learning rate 3×10^{-4}, discount $\gamma = 0.99$, clip range $\epsilon = 0.2$, 4 probe batches for rewards, and no penalties ($\lambda_g = \lambda_s = 0$).

4.2 Main Results

Table 1 shows validation performance and training loss at step 561 (end of training).

JEL achieves a validation BPB of 1.173 at step 561, improving 2.5% over schedule-free (1.203) at equivalent cost and outperforming the suboptimal baseline (1.453) by 19.3%. Since JEL and the suboptimal baseline share identical base hyperparameters (the only difference is the controller being active vs. disabled), this gap directly measures the controller's contribution. JEL reaches within 8.4% of the hand-tuned upper bound (1.082). Figure 2 shows training dynamics; JEL consistently improves over schedule-free while approaching hand-tuned performance.

Table 1: Comparison of validation performance (bits-per-byte) and training loss at step 561 (end of training). Lower is better for both metrics.

Method	Val BPB (561)	Train Loss (561)
Schedule-Free Baseline	1.203	3.932
JEL	**1.173**	3.810
Hand-Tuned Upper Bound	1.082	3.507
Suboptimal Baseline	1.453	4.718

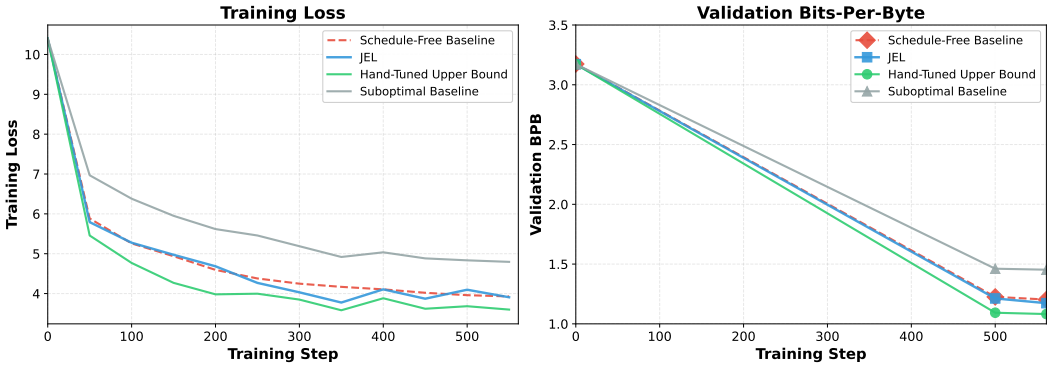

Figure 2: Training dynamics across methods. JEL improves over the schedule-free baseline while approaching the hand-tuned upper bound from extensive manual experimentation.

4.3 Computational Cost Analysis

Traditional ASHA sweeps require $\sim$6 full training runs, while our schedule-free baseline required 8 runs. JEL's meta-training runs 400 total episodes on the full 73.5M-parameter architecture at reduced sequence length (128 tokens), reducing per-episode cost by $\sim 16\times$. On the same 8-GPU setup, meta-training completes in $\sim$5.6 run equivalents (measured wall-clock), comparable to ASHA but with full reusability. After initial training, JEL requires only $1\times$ base cost versus 6–8$\times$ for repeated sweeps. Deployment adds negligible overhead ($<1\%$ from controller inference every 50 steps).

4.4 Ablations and Negative Results

We explored richer controller designs that consistently degraded performance. Adding gradient norms, loss values, or momentum statistics to the state reduced final performance, suggesting the reward signal provides sufficient optimization feedback. Including gradient or smoothness penalties ($\lambda_g, \lambda_s > 0$) and expanding to per-optimizer control (4D actions) also hurt results. Standard GRPO with length and variance normalization caused entropy collapse within 10 updates, preventing exploration; removing these normalizations maintained stable entropy ($\approx 0.3 - 0.5$) essential for learning effective policies. These results suggest JEL's effectiveness comes from simplicity: minimal state, direct reward signal, and stable training rather than complex feature engineering.

5 Discussion

5.1 When Does JEL Help and Why Minimal State Works

JEL improves 2.5% over schedule-free optimization and 19% over suboptimal baselines, demonstrating that an RL-optimized hyperparameter schedule, learned from hundreds of training episodes, outperforms both fixed schedules and schedule-free approaches while adapting hyperparameters rather than matching defaults. JEL reaches within 8.4% of hand-tuned upper bounds from extensive manual experimentation. This gap likely reflects that hand-tuning incorporates knowledge of model architecture and dataset properties difficult to encode in minimal state, though the tradeoff favors JEL's automation especially when training many related models.

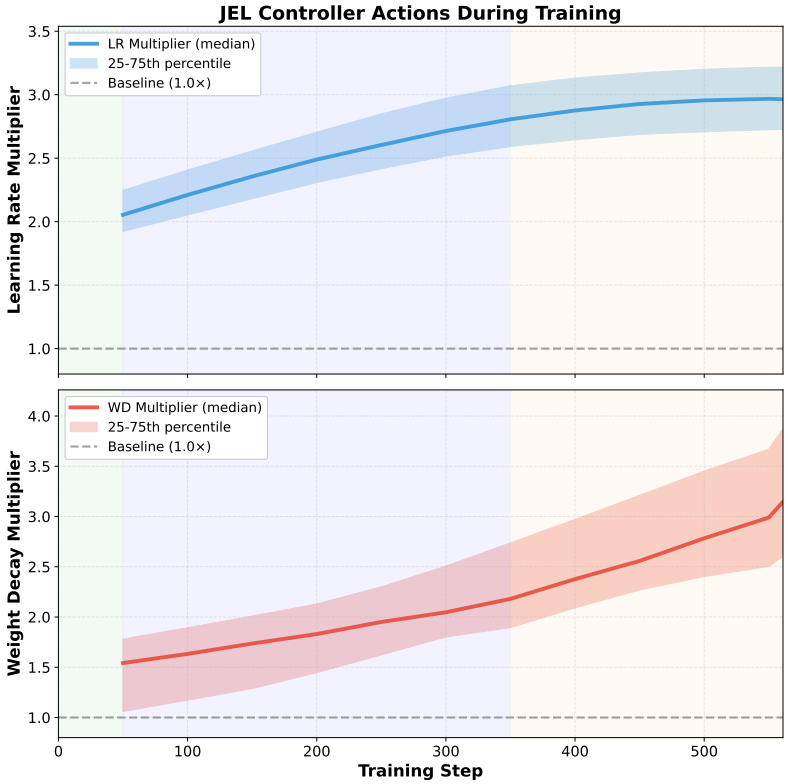

Figure 3: JEL controller actions during training. The learned policy progressively increases both multipliers throughout training, compensating for the conservative base hyperparameters (LR 10^{-3}, WD 10^{-2}) by scaling them toward more aggressive values as training progresses. Solid lines show median multipliers at each decision point across 50 meta-training updates, with shaded regions indicating 25-75th percentiles.

Notably, a 1D state (training progress only) outperformed all richer representations we tested, including gradient norms, loss values, and momentum statistics. We hypothesize the reward function already provides rich feedback about optimization quality, making instantaneous statistics redundant. Additionally, richer states may cause overfitting during meta-training; with only 400 training episodes, complex representations may not generalize well. The minimal state forces learning general trajectory-level patterns transferring across initializations and data orderings.

5.2 Training Stability

Removing length and variance normalizations from GRPO proved essential. Standard normalizations caused entropy collapse within 10 updates as the policy converged to deterministic actions. Removing these while retaining group-relative baselines maintained exploration (entropy 0.3-0.5) throughout training, critical for discovering effective hyperparameter trajectories.

5.3 Limitations and Future Work

Transfer across scales. We evaluated on 73M parameter models; testing transfer to larger models (350M-1B parameters) is important, though the multiplicative parameterization should aid scale invariance. **Per-optimizer control.** We use unified multipliers for simplicity, but preliminary experiments on simpler architectures suggest separate per-optimizer control may close the remaining gap to hand-tuned baselines. **Longer training.** Our 561-step evaluation is short; many practical settings involve 10K-100K+ steps where the optimality gap may differ.

6 Conclusion

We presented JEL, a learned hyperparameter controller that adjusts learning rate and weight decay during training via schedules distilled from reinforcement learning. By framing training as an episodic RL problem with modified GRPO, we train a compact policy that captures effective hyperparameter trajectories without per-task sweeps.

On transformer pretraining, JEL improves 2.5% over schedule-free optimizers at equivalent cost, reaching within 8% of hand-tuned upper bounds. Controller training costs 5.6 run equivalents, less than both ASHA ($\sim$6 runs) and schedule-free sweeps (8 runs), while producing a reusable policy. Key findings: (1) minimal state (training progress only) outperforms richer observations, (2) removing GRPO normalizations prevents entropy collapse, (3) direct loss improvement provides sufficient reward signal.

JEL demonstrates that learned hyperparameter control can reduce optimization costs in deep learning, offering a practical alternative to expensive searches with particular value in resource-constrained settings or when training many related models.

Reproducibility Statement

All hyperparameters for both controller training and base model training are listed in Table 2. The controller architecture is fully specified in Appendix A.1. Computational details including hardware, throughput, and overhead measurements are reported in Appendix A.

References

Aaron Defazio, Konstantin Mishchenko, and Ashok Jagiello. The road less scheduled. *arXiv preprint arXiv:2405.15682*, 2024.

Bach Do and Ruda Zhang. Multi-fidelity bayesian optimization: A review. *arXiv preprint arXiv:2311.13050*, 2023. Published in AIAA Journal 63:6 (2025) 2286-2322.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pp. 1437–1446. PMLR, 2018.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Benjamin Thérien, Charles-Étienne Joseph, Boris Knyazev, Edouard Oyallon, Irina Rish, and Eugene Belilovsky. μlo: Compute-efficient meta-generalization of learned optimizers. *arXiv preprint arXiv:2406.00153*, 2024.

Zhen Xu, Andrew M. Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule. *arXiv preprint arXiv:1909.09712*, 2019.

A Appendix

A.1 Implementation Details

Controller architecture. The policy network is a 2-layer feedforward network with architecture:

```
state (1D) -> Linear(1, 64) -> Tanh
           -> Linear(64, 64) -> Tanh
           -> Linear(64, 4) -> (mean_lr, mean_wd, log_std_lr, log_std_wd)
```

Log standard deviations are clamped to $[-5, 2]$ to prevent numerical instability. Initial log_std bias is set to -0.5, corresponding to $\sigma \approx 0.61$.

Hyperparameter values. Table 2 lists all hyperparameters used in our experiments.

Table 2: Complete hyperparameter configuration for JEL controller training and deployment.

Category	Parameter	Value
Controller	Hidden dimension	64
	Policy learning rate	3×10^{-4}
	Initial log std	-0.5
	Action range	$[\log(0.01), \log(100)]$
PPO/GRPO	Discount factor γ	0.99
	Clip range ϵ	0.2
	Entropy coefficient	0.0 (ablated)
	PPO epochs per update	4
	Minibatch size	64
Training	Decision interval Δ	50 steps
	Episode length	561 steps
	Updates	50
	Episodes per update	8 (1 per GPU)
Reward	Reward batches	4
	Gradient penalty λ_g	0.0
	Smoothness penalty λ_s	0.0
Base model	Layers	6
	Model dimension	384
	Attention heads	3
	Batch size, deployment (tokens)	524,288
	Batch size, meta-training (tokens/device)	4,096
	Total training steps	561

A.2 Additional Experimental Results

Schedule-free sweep details. Table 3 shows all configurations tested in the schedule-free baseline sweep.

Table 3: Schedule-free optimizer sweep results. All configurations use Muon LR=0.02 and Muon WD=0.2.

Config	Schedule-free LR	WD	Warmup	Val BPB	Time (min)
0	0.5	0.00	0	1.456	9.10
1	0.5	0.01	0	1.453	9.08
2	1.0	0.00	0	1.347	9.10
3	1.0	0.01	0	1.348	9.10
4	2.0	0.00	0	**1.203**	9.10
5	2.0	0.01	0	1.291	9.11
6	1.0	0.01	100	1.396	9.07
7	1.0	0.01	300	1.429	9.07

A.3 Computational Environment

All experiments were conducted on NVIDIA A100-SXM4-40GB GPUs. Distributed training used 8 GPUs with PyTorch DDP. Per-step training time averaged 380ms, achieving approximately 1.38M tokens/second throughput (9.8% model FLOP utilization). Controller inference added $<$0.5ms overhead per decision point (50 steps), representing $<$0.3% additional cost.