EVOFORGE: AGENTIC ONLINE HYPERPARAMETER TUNING FOR PRETRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Pretraining hyperparameters are typically fixed before training begins, while the optimal configuration drifts as training proceeds. Existing automated methods, such as random search, Bayesian optimization, multi-fidelity scheduling, and population-based training, presuppose multiple trials or concurrent replicas; a single large-scale pretraining run yields one trajectory, precluding a second trial. We present EvoForge, to the best of our knowledge, the first system that places LLM agents in the control loop of a running distributed pretraining job. We cast online tuning as a single-trajectory, partially observable sequential decision problem, where observations are induced by past actions, and organize the system around three cooperating agents, namely, one for tuning, one for safety, and one for cross-task reflection. EvoForge distills rules into a two-layer experience store only after sufficient evidence has accumulated. In GPT-2 124M trained on 10B FineWeb-Edu tokens, EvoForge reduces the final validation loss by 4.15% relative to a tuned cosine schedule.

1 INTRODUCTION

Pretraining a language model typically begins with a configuration fixed before the first gradient step, including the learning-rate schedule, batch configuration, weight decay, and optimizer coefficients. Yet the configuration that best suits the run is not stationary. As the model evolves, so do gradient noise and curvature, causing the critical batch size to increase over training (McCandlish et al., 2018; Shallue et al., 2019). Likewise, hyperparameters that are effective in the early stages may become suboptimal near the end of the training budget (Smith et al., 2018; Hu et al., 2024). **A single large-scale run offers only one trajectory, so a mismatched configuration must be revised online, and since no run of this scale can be watched by hand, that revision has to be made by the system itself; a failed revision must be recovered from** (Radford et al., 2019; Yang et al., 2025).

Existing automation does not answer this. Random search and Bayesian optimization (Bergstra & Bengio, 2012; Snoek et al., 2012), multi-fidelity scheduling (Li et al., 2018; 2020), population-based training (Jaderberg et al., 2017), and differentiable hyperparameter optimization (Maclaurin et al., 2015) rank configurations by comparing evaluated trajectories, so each presupposes multiple trials or concurrent replicas: one trajectory leaves nothing to compare against, nothing to prune or copy, and no second trial to revisit a failed configuration (Figure 1). LLMs have been used as black-box optimizers (Yang et al., 2024) and as agents for software and scientific tasks, but the closest proposals recommend hyperparameters across trials rather than within a run, where a proposal costs a separate run and a poor one is discarded rather than undone (Liu et al., 2024); a model placed inside a running job acts within a bounded, schema-validated action space, in a single, non-distributed run (Rao & Advani, 2026). Interactive training frameworks make the loop externally addressable but stop at validating a request before it is applied, leaving detection-triggered recovery and anything retained across tasks to the user (Zhang et al., 2025b; 2026).

Moving the choice inside the loop is therefore not a matter of execution, since rewriting a learning rate at a step boundary is trivial, but of decision, and three difficulties follow. The signal that would justify a change is still forming when it must be made, and the observables are induced by past actions, so a controller can manufacture the evidence it later reasons from. An error once taken hold cannot be withdrawn, and a guardrail can only stop a proposal that has not yet taken effect.

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.

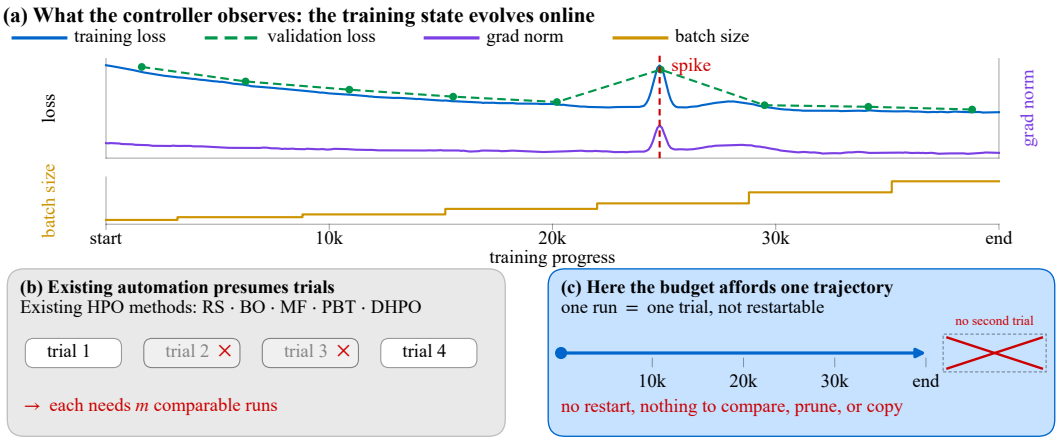

Figure 1: Motivation. (a) The signal the controller observes while the job runs: the training loss, the validation loss, the gradient norm, and the global batch size that the controller raises; all vary step to step and occasionally carry an anomaly, such as the spike marked here. (b) Existing hyperparameter optimization (HPO) methods, random search (RS), Bayesian optimization (BO), multi-fidelity scheduling (MF), population-based training (PBT), and differentiable hyperparameter optimization (DHPO), rank configurations by comparing trials, and so presuppose multiple runs. (c) One large-scale run affords a single trajectory: the problem is not a search over configurations under a finite trial budget.

Experience is perishable and uneven: a judgment not retained does not transfer to the next task, while not every judgment deserves to.

We present EvoForge, to our knowledge the first system to place LLM agents in the control loop of a running distributed pretraining job. Decisions take effect at step boundaries, already synchronized across data-parallel replicas, so an intervention reaches every replica identically. EvoForge separates the three difficulties into a tuning agent, a safety agent, and a cross-task experience distillation and reflection agent, sharing one view of the training state and one two-layer experience store.

In GPT-2 124M trained on 10B tokens of FineWeb-Edu, EvoForge reduces the final validation loss by 4.15% relative to a tuned cosine schedule and by 2.49% relative to the strongest static baseline, WSD, with the zero-shot scores moving in the same direction.

Our contributions are as follows.

- We introduce **a single-trajectory, partially observable formulation of online tuning**, whose observables are induced by past actions, and separate the three problems any solution must answer: a biased signal, an irreversible action, and perishable experience (§3).
- We propose **EvoForge**, a system of three cooperating agents over one two-layer experience store, for online tuning, guarded and reversible intervention, and cross-task experience distillation (§4).
- We conduct **controlled experiments** against three static schedules under a matched 10B-token budget, with ablations, a larger scale, a mixture-of-experts model, and a different controller backend, demonstrating that the gain does not rest on a single schedule, scale, or controller model (§5).

2 Related Work

Hyperparameter optimization. Existing hyperparameter optimization methods rank configurations by comparing distinct evaluation trajectories: random search and Bayesian optimization sample the configuration space under different acquisition strategies (Bergstra & Bengio, 2012; Snoek et al., 2012); Hyperband and ASHA allocate budget across many low-fidelity trials and successively

prune weaker ones (Li et al., 2018; 2020); population-based training copies well-performing workers and mutates their hyperparameters mid-run (Jaderberg et al., 2017); differentiable hyperparameter optimization backpropagates hyperparameter gradients through the training procedure (Maclaurin et al., 2015). All presuppose multiple trials or concurrently running workers; the nearest relative, population-based training, still needs a population of parallel runs. The premise fails when the compute budget sustains a single trajectory, as in large-scale pretraining: there is no generation to compare against, nothing to prune or copy, and no second trial in which a failed configuration could be revisited. Our problem is thus not to search for the best configuration under a limited trial budget, but to decide, on one ongoing trajectory, whether to change, what to change, and how to recover once a change fails.

LLMs as optimizers and agents. LLMs have been used as black-box optimizers (Yang et al., 2024), as tools for algorithm discovery, and as agents for software and scientific tasks (Yao et al., 2023; Shinn et al., 2023; Park et al., 2023; Hong et al., 2024; Zhang et al., 2025a). The works closest to ours use an LLM to propose hyperparameters, but across trials rather than within a run: the model reads a record of completed experiments and emits the configuration for the next trial (Yang et al., 2024; Liu et al., 2024), so a proposal is paid for by a separate run and a poor one is discarded rather than undone. Where an LLM has instead been placed inside a running job, it stays within a schema-validated, bounded action space in a single, non-distributed run, with no hard safety guarantees reported (Rao & Advani, 2026). To our knowledge, no prior work puts an LLM in the control loop of a distributed pretraining job—where an intervention must take effect identically across data-parallel replicas—nor treats the failure modes that arise there.

Interactive training frameworks. Interactive Training (Zhang et al., 2025b) and its successor (Zhang et al., 2026) make the training loop externally addressable: a control server, later generalized into a shared protocol with an audit journal, lets humans or automated controllers submit requests that the training loop validates and applies at safe control points. These frameworks supply the interface EvoForge builds on, but they stop at it: validation covers declared types, ranges, and action legality—the authors state that such bounds "cannot guarantee safe or optimal actions" and list approval gates, rollback policies, and task-specific safety checks as still missing—recovery is user-initiated, nothing is retained across tasks, and neither work addresses distributed execution. Our contribution is the layer above that interface: constraints on training-dynamics risks, an admission gate, detection-triggered recovery with state-consistent rollback, and a cross-task experience store.

3 Problem Modeling

Moving the choice of hyperparameters from before training into training itself asks the controller to hold three things at once: observations it can trust, changes it can apply safely, and judgments it can accumulate across tasks. These constrain one another—the observables are induced by past actions, and an action's consequences in turn change the values later observables take—so we fix the setting and the representation first (Appendix A), then the decision problem and the three problems any solution must answer (§3).

Pretraining minimizes the validation loss under a fixed token budget. Writing $B_t = \mathrm{GBS}_t$ for the global batch size, the update at step t is

$$\theta_{t+1} = \theta_t - \eta_t \cdot \mathrm{clip}\Big(\tfrac{1}{B_t}\textstyle\sum_{i=1}^{B_t} \nabla \ell_i(\theta_t),\, c\Big), \tag{1}$$

with η_t the effective learning rate and c the gradient-clipping norm; training uses fused AdamW (Loshchilov & Hutter, 2019; Kingma & Ba, 2015) in mixed precision (Micikevicius et al., 2018), and the batch gradient is produced by W data-parallel replicas over N micro-batches of size b (Wolf et al., 2020), so that a step consumes GBS_t samples through the per-replica micro-batch size MBS_t and the accumulation steps $\mathrm{GAS}_t = \mathrm{GBS}_t/(\mathrm{MBS}_t \cdot W)$—three coupled runtime quantities rather than independent hyperparameters (Appendix A).

We model online tuning as a single-trajectory, partially observable sequential decision problem. At observation time t the agent observes

$$s_t = \big(\{\ell_{t-i}\}, \{v_{t-i}\}, \{\eta_{t-i}\}, \{\|\hat{G}_{t-i}\|\}, \{B_{\text{simple},t-i}\}, \text{mem}_t, \text{thr}_t, \text{cfg}_t, H_t\big), \tag{2}$$

the loss, learning-rate, gradient-norm and critical-batch histories (the latter following McCandlish et al., 2018), memory and throughput, retrieved experience, the current configuration, and its own decision history. It selects an action $a_t \in \{\texttt{keep}, \texttt{update}, \texttt{rollback}\}$, applied by an executor at the step boundary with consequences visible only indirectly in later losses, and minimizes the final validation loss $\mathcal{L}_{\text{val}}(\theta_T)$ at a fixed token budget $\sum_t \text{GBS}_t = \text{budget}$.

One structural difference separates this from classical hyperparameter optimization (HPO) (Bergstra & Bengio, 2012; Snoek et al., 2012; Li et al., 2018; Jaderberg et al., 2017): the outcome of an HPO trial does not depend on how its configuration was chosen, whereas here the observables are functions of the current model state, itself the result of past actions, so observations and actions are joined by a feedback path and the problem is not a contextual bandit. An HPO trial ends and is scored, so information is complete and a bad configuration is discarded rather than undone; here a single trial cannot be restarted, information is a truncated projection of one trajectory contaminated by past actions, feedback takes tens to thousands of steps, and an error is irreversible because the compute is already spent.

A concrete consequence appears in batch control: the critical-batch estimate $B_{\text{simple}} = \hat{S}/\|\hat{G}\|^2$ is not independent of the learning rate—its denominator depends on where the optimizer sits on the loss surface, so lowering the rate raises it while the useful batch is unchanged, and a controller that pushes the batch up on that basis reinforces itself. The estimator is reliable offline; only the closed loop creates the risk, handled by the validation layer of §4.2.

Three problems follow. (P1) Decide continuously from a biased, endogenous signal: judgments must be made while information is still forming, and the observables are induced by past actions, so the controller can manufacture the evidence it reasons from. **(P2) Keep an irreversible action reversible**: the compute is already spent by the time a mistake becomes visible, and a guardrail can only stop a proposal that has not yet taken effect. **(P3) Accumulate judgments across tasks**: experience is perishable unless explicitly retained, and not all of it deserves to guide the next task. The three differ in time scale and failure mode; §4 separates decision, recovery, and experience inheritance accordingly and answers **(P1)**–**(P3)** in turn.

4 METHOD

The three problems of §3 differ in time scale and failure mode, and the solution separates accordingly. EvoForge runs as a separate process beside the distributed training job, consuming metrics over WebSocket and issuing commands over HTTP (Figure 2), with decisions computed in a background thread so that training never blocks on LLM inference; the three agents share one view of the training state and one experience store. **(P1)** is answered by the **tuning agent** (§4.1), which carries the action of the moment and conditions its judgment on phase and history; **(P2)** by the **safety agent** (§4.2), which prevents errors and reverses those that take effect; **(P3)** by the **cross-task experience distillation and reflection agent** together with the experience store it writes to (§4.3). Their trigger times do not overlap—the tuning agent fires on the observation window it sets itself, the reflection agent at the end of a task—yielding a fast within-task control loop and a slow cross-task loop that closes once per task; the main loop is Algorithm 1 and implementation detail is in Appendices A–E.

4.1 THE TUNING AGENT

The tuning agent is the only agent that acts under normal conditions: it emits a fixed-schema structured proposal, and the rest of the system decides whether and how to execute it. How reliable that proposal is depends on what the prompt contains and on when the model is asked again, which fixes four design choices: **recency-biased downsampling** sets the resolution at which the trajectory enters the prompt, **phase-aware guidance** injects the experience that matches the current phase, **reviewing its own decisions** reads back the model's own judgments and how they were handled, and a **self-paced observation window** ties the control frequency to the magnitude of the last change.

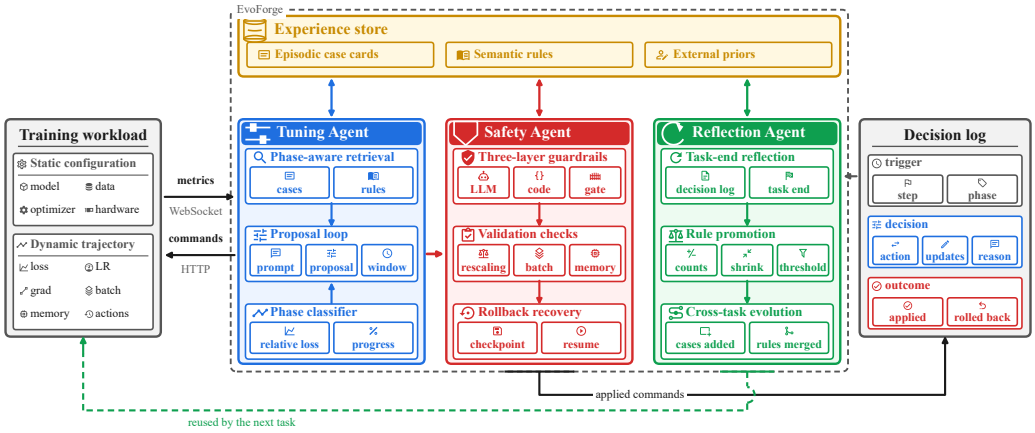

Figure 2: Architecture. EvoForge runs as a separate process beside the training job (left), observed and commanded over WebSocket. Its workload representation separates static configuration fields from dynamic trajectory signals. The tuning agent sets its observation window, retrieves phase-aware guidance from the two-layer experience store, and emits a fixed-schema proposal; the safety agent checks it at proposal, validation, and admission, choosing the restore point and post-rollback hyperparameters on rejection or drift. Decisions are logged (right); at task end the reflection agent distils them into case cards and rules promoted by support and opposition counts.

Recency-biased downsampling. Trajectories have no bounded length, whereas a single judgment consumes a bounded amount of information: the whole loss curve would exceed the prompt budget, and uniform sampling would let the recent trend be diluted by distant history. The curve is therefore mapped onto the time axis by a power law with an exponent greater than one, taking a fixed number of points so that more of them fall on recent steps. The number of points is fixed, so the input size does not grow as training proceeds, while the resolution of the latest trend is preserved.

Phase-aware guidance. The same change may be ordinary exploration early in training and an error during the closing anneal, so the prompt carries a phase label together with the behaviour that phase implies, aligning the judgment with the current phase. The plateau is where control is most easily lost: interventions repeatedly raise and lower the learning rate until they become a self-oscillation, so its guidance text names that failure mode outright, and batch policy is deliberately exempt from this conservatism because growing the batch is a throughput decision rather than a corrective one. The guidance comes from the experience store rather than from a fixed rubric: the phase decides which class of experience to retrieve. How the phase is judged are in Appendix B.

Decision review. The model cannot tell from a single prompt how the system disposed of its earlier proposals, so it may repeat a change that was already refused or that eventually led to a rollback. The history block therefore reads back recent decision records, including blocked and rolled-back ones, each with its rationale and confidence. This also gives it evidence about its own calibration, on which it adjusts the confidence it reports next.

Self-paced observation window. A fixed control interval is inherently mistuned: after a large change it is too short, tempting the agent to judge before the change has taken effect, which is exactly where oscillation comes from; in a calm period it is too long, delaying an adjustment that could have been made. Each decision therefore carries a minimum number of steps to observe, clamped between hard bounds, so that the control frequency follows the magnitude of the change just applied and both mismatches disappear.

4.2 The Safety Agent

Errors from online intervention are irreversible, so the system cannot assume that proposals are correct. Safety has two tiers: **three-layer guardrails** keep erroneous proposals from taking effect,

and **rollback recovery** undoes the errors that do. Prevention alone has no fallback when a guardrail misjudges; recovery alone makes training pay the price repeatedly.

Three-layer guardrails. We treat the LLM as a useful but unreliable proposer: proposals come from the model, while validation and admission are enforced by deterministic code(Table 4 in Appendix D). Validation *explains rather than corrects*: a rejected proposal returns to the model with the reason and the acceptable range, because silently applying a modified version would equally contaminate the history the model reasons from. Memory safety is *predictive rather than reactive*: the largest safe micro-batch is reported during prompt construction, and proposals with insufficient predicted headroom are rejected, since waiting for a real out-of-memory error would interrupt the job. Admission applies a proposal only above the confidence threshold, and blocked proposals stay visible in the decision history. Three further constraints on batch-related proposals correspond to the endogeneity identified in §3; their criteria and rationale are in Appendix D, and the observability bound on the accumulation steps in Appendix C.

Rollback recovery. When the tuning agent judges that training has seriously deteriorated (the loss has jumped and is not recovering, a change has clearly degraded training, or continuing would waste the budget), it emits `rollback` with a symptom description and a suspected starting step; because a rollback discards every step after the restore point, it is gated on a higher confidence threshold and a cooldown between rollbacks, which prevents a rollback-retry-rollback oscillation. The safety agent then decides which checkpoint to restore (the most recent one before the bad change, since retreating further only enlarges the loss) and which hyperparameters to resume with, usually undoing the change that caused the divergence. A rollback is not a local operation: once the training side's global step moves backwards, the agent's own history must be truncated to the restore point, and post-rollback hyperparameters must wait until training passes that point (Appendix D). If the LLM is unavailable or its plan unusable, the system reverts to a deterministic undo of the most recent effective change.

4.3 Cross-Task Experience Distillation and the Experience Store

The phase guidance of §4.1 comes from distilled experience, and the difficulty is maintaining it: as long as it stays manual (review an unstable run, hand-edit the prompt), accumulation is bounded by human effort and covers few conditions. The reflection agent automates the process (Shinn et al., 2023), reading the full decision log and trajectory at the end of every task and producing case cards worth reusing together with the rules this task's evidence supports or contradicts; its output enters the experience store, which externalizes tuning knowledge into retrievable, updatable items. This fixes three parts: the **knowledge store** determines in what form knowledge is kept and where it comes from, **retrieval and injection** decides which items enter the prompt, and **cross-task evolution and the closed loop** decides how the store is updated as the system runs.

Knowledge store. The store borrows the distinction between episodic and semantic memory (Park et al., 2023) and keeps two complementary forms: **case cards** record a single event (situation, action, outcome, and a lesson for future decisions), while **rules** are guidance statements that enter a prompt directly, each with a support count s and an opposition count c. The two differ in testability: case cards are traceable but generalize weakly, rules guide behaviour directly but may overfit small samples, so they supply evidence and constraints respectively. Rules are deliberately not limited to the system's own runs: besides rules induced from decision trajectories, practical experience entered by hand and conclusions drawn from papers or technical reports can be entered as items. The system has run under few conditions, so rules induced from its own experience alone would leave every unencountered condition without a prior; and the evidence for an external conclusion resides at its source rather than in the system, so the confidence of an external rule is assigned by whoever entered it and takes no part in counting and promotion, the two classes of source being governed separately. Item fields and the structure of the store are in Appendix E.

Retrieval and injection. The tuning agent retrieves by the similarity between the current phase and the situation description of Appendix A, and injects the most relevant case cards together with the rules applicable to that phase. The candidate set may be large while genuinely comparable experience is rare, so matching proceeds in three layers: phase and model agreement fixes the candidate set; outcome severity ranks within it, because failures, rollbacks, and successes are all more informative than neutral cases; and cross-scale experience is down-weighted, to avoid misapplying

lessons learned at 124M to larger models (Yang et al., 2022). Only rules with sufficient evidence are injected: each induced rule keeps its two counts and takes a Beta(1,1) shrinkage confidence

$$\text{confidence} = \frac{s+1}{s+c+2}, \tag{3}$$

entering the prompt as stable guidance only once its support reaches a lower bound and its confidence exceeds a promotion threshold, since the shrinkage prior suppresses small-sample exaggeration and a single anecdote therefore falls well short. Rules entered by hand or drawn from the literature join injection at the confidence they were given. The retrieval algorithm and the details of the three-layer matching are in Appendix E.

Cross-task evolution and the closed loop. The store keeps evolving as the system runs, since neither its initial priors nor the evidence it holds cover the conditions still to come: when a task ends, case cards are appended; rules are compared against existing items, merged with accumulated counts when semantically equivalent, and entered with initial counts when new; rules below the bar keep accumulating evidence, and a rule contradicted by later evidence is downgraded as its opposition count rises. This is the slow loop of §4: a task ends and produces case cards and rules; rules accumulate evidence and become eligible for promotion; promoted rules and their case cards are retrieved at the corresponding phase of the next task, changing its decisions and producing new evidence. Closing once per task, its effect appears over a sequence rather than in a single run; the behaviour of these mechanisms and the evidence behind each threshold are reported in §5.

5 Experiments

The experiments answer three questions. **Q1**: under the same token budget, does online tuning improve model quality over well-tuned static schedules (§5.2)? **Q2**: how much does each agent contribute to the gain (§5.3)? **Q3**: does the improvement persist at a larger scale and carry over to a mixture-of-experts architecture (§5.4)?

5.1 Experimental Setup

The main experiments use GPT-2 124M (Radford et al., 2019) at sequence length 1024, trained on 10B tokens of FineWeb-Edu (Penedo et al., 2024). The baselines are three static schedules: tuned cosine, WSD (Hu et al., 2024), and the nanoGPT reference implementation (Karpathy, 2022), differing only in schedule shape. EvoForge uses a commercial LLM as the controller at temperature 0.3 and a confidence threshold of 0.7. The primary metric is the final validation loss, complemented by zero-shot accuracy on five downstream tasks, ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), and COPA (Roemmele et al., 2011). The full training configuration, per-baseline hyperparameters, and evaluation protocols are in Appendix F.

5.2 Comparison with Static Baselines

Method	Val. loss ↓	ARC-Easy	HellaSwag	PIQA	SciQ	COPA	Avg.
cosine (static)	3.2530	45.50	29.94	60.88	66.90	62.00	53.04
WSD (static)	3.1976	46.09	30.36	**62.13**	67.90	64.00	54.10
nanoGPT (static)	3.2465	45.03	30.14	61.53	66.50	62.00	53.04
EvoForge	**3.1180**	**47.26**	**30.80**	<u>61.92</u>	**69.20**	**67.00**	**55.24**

Table 1: Final validation loss and downstream scores under a matched 10B-token budget. All runs consume the same budget; the static baselines run at $\mathrm{GBS} = 512$, $\mathrm{MBS} = 32$ throughout, while EvoForge starts from $\mathrm{GBS} = 64$, $\mathrm{MBS} = 4$ and raises the batch during training.

Table 1 answers Q1. EvoForge attains the lowest final validation loss of the four runs, 3.1180, 4.15% below the tuned cosine schedule and 2.49% below the strongest baseline, WSD. The downstream

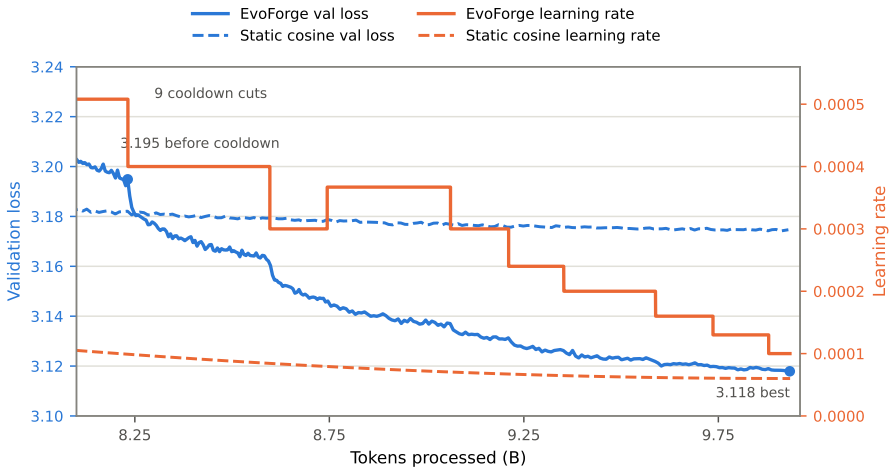

Figure 3: Late-phase annealing between 8B and 10B tokens. Validation loss of EvoForge and of the static cosine baseline on the left axis, with the learning rate on the right.

Configuration	Val. loss ↓	ARC-Easy	HellaSwag	PIQA	SciQ	COPA	Avg
EvoForge (Full system)	**3.1180**	**47.26**	**30.80**	**61.92**	**69.20**	**67.00**	**55.24**
w/o reflection agent	3.2279 (+3.5%)	44.19	29.16	60.39	65.40	64.00	52.63
w/o safety agent	3.1513 (+1.1%)	46.30	29.78	61.64	67.70	63.00	53.68

Table 2: Ablations. All configurations share the same starting configuration, token budget, controller LLM, and confidence threshold as the full system. "+X%" denotes the relative increase in validation loss over the full system. Best value in each column is in **bold**.

scores follow the same ordering: EvoForge has the highest mean over the five tasks, 55.24 against 53.04 to 54.10 for the baselines.

Figure 3 shows one reason the loss ends lower. Over the last two billion tokens, nine cooldown cuts lower the loss from 3.195 to 3.118, while the baseline stays flat at 3.176 across the same window: part of the final gap was therefore opened in the closing phase, not only earlier in the run. Appendix G gives the full decision trajectory and tracks the downstream scores every billion tokens.

5.3 Ablations

Table 2 answers Q2. It removes each agent in turn, under the same starting configuration, token budget, controller LLM, and confidence threshold, and reads each agent's contribution as the degradation its removal causes: 0.1099 in validation loss (+3.5%) for the reflection agent against 0.0333 (+1.1%) for the safety agent. Both removals raise the loss and lower every downstream score, so the two contributions are distinct rather than redundant.

Removing the cross-task experience distillation and reflection agent raises the final validation loss from 3.1180 to 3.2279 (+3.5%), back into the band occupied by the static baselines, and lowers the five-task mean from 55.24 to 52.63. The experience layer is not confined to the cross-task loop: retrieval matches case cards and rules by phase and situation similarity, so the priors held in the store enter the prompt even with a single task in play, which is the condition of this ablation.

Removing the safety agent removes the guardrails, the detection stage, and recovery together, so a proposal takes effect at the next step boundary unvalidated and cannot be undone. The layer intervened rarely (one rejection by the guardrails, four to five withholdings by the gate), and what its absence costs is the error that recovery undid: the learning-rate probe at 6.7B, worth 0.028 in loss, stays in the trajectory, the validation loss rises from 3.1180 to 3.1513 (+1.1%), and the five-task mean falls from 55.24 to 53.68. The cost remains well below the +3.5% of removing the reflection

agent, and structurally so: the experience layer shapes every decision, whereas the safety layer acts only when something has already gone wrong.

5.4 Scaling: Model Size and Architecture

Table 3 answers Q3. It repeats the comparison on two models beyond the 124M setting, Qwen3-0.6B (Yang et al., 2025) and a mixture-of-experts model with 0.7B total parameters. Under the same 10B-token budget, the final validation loss falls by 4.15% on GPT-2 124M, by 15.63% on Qwen3-0.6B (3.252 to 2.744), and by 15.12% on Qwen3-MoE-Tiny (3.353 to 2.846). The gain is therefore not confined to a single scale, nor to a dense architecture: it persists on both new models, and there it is larger than on the smallest one. The downstream columns agree in every cell. Across the fifteen model-task comparisons, EvoForge is higher than its baseline in all fifteen, with an average gain of about 2.2 points on GPT-2, 8.8 points on Qwen3-0.6B, and 8.2 points on the MoE model. A further experiment, which swaps the controller backend under a fixed starting configuration, is in Appendix H.

Model	Method	Val. loss ↓	ARC-Easy	HellaSwag	PIQA	SciQ	COPA
GPT-2 (124M)	Baseline	3.253	45.50	29.97	60.88	66.90	62.00
	EvoForge	**3.118** (-4.15%)	**47.26**	**30.80**	**61.92**	**69.20**	**67.00**
Qwen3 (0.6B)	Baseline	3.252	40.78	28.79	59.85	59.10	60.00
	EvoForge	**2.744** (-15.63%)	**53.66**	**39.89**	**64.64**	**70.40**	**64.00**
Qwen3-MoE-Tiny (0.7B)	Baseline	3.353	39.65	27.40	56.86	57.20	62.00
	EvoForge	**2.846** (-15.12%)	**48.40**	**36.13**	**64.15**	**69.50**	**66.00**

Table 3: Validation loss and downstream task accuracy. For Val. loss, the parenthesized value is the relative reduction of EvoForge over the baseline. Downstream values are accuracy (%). Best results in each column are in **bold**.

6 Limitations

Three boundaries qualify these results. First, every run fits on a single eight-GPU node, so training across multiple nodes remains untested. Second, the largest model we train has 0.7B parameters and every run consumes 10B tokens, far below frontier scale, so whether the gain survives much larger models or corpora is untested as well. Third, the parallelization strategy stays outside the online decision: tensor, pipeline, and sharded degrees are fixed, and every run is 8-way data parallel on one node (Appendix F). We target the final validation loss under a matched budget, not the time to reach it, so whether a given loss arrives sooner remains open.

7 Conclusion

We cast online tuning as a single-trajectory, partially observable decision problem and presented EvoForge, which places LLM agents in the control loop of a running distributed pretraining job. A tuning agent decides what to change, a safety agent bounds each decision before it takes effect and can undo it afterwards, and a reflection agent distills what recurs into a two-layer experience store for later tasks. EvoForge lowers the final validation loss by 4.15% relative to a tuned cosine schedule and by 2.49% relative to the strongest static baseline, WSD, with zero-shot scores moving in the same direction, and the gain reappears at a larger scale and on a mixture of experts. More broadly, what an automated trainer needs is not a better search over configurations but a decision it can revise and undo, and experience that outlives the run that produced it.

AI use statement

In this work, our use of generative AI tools, as declared in the submission form, falls into three categories. *Assistance with writing*: the tools helped draft and edit parts of the manuscript and translated material between Chinese and English. *Retrieval and discovery*: they were used to search for

and summarize related literature, and to check background facts about the baselines and the tooling we compare against. *Ideation and execution*: we used them to brainstorm candidate mechanisms, expected failure modes, ablations, and hyperparameter ranges, and to write and debug the support scripts that produce the figures and the evaluation tables.

Everything that enters this paper as a claim or a number is ours. We did not use these tools to generate training or evaluation data, or to derive the formulation and the mathematical statements of §3 and Appendix C. The design decisions behind the experiments, the training-side implementation and the reading of the results are our own; AI-assisted code is confined to the support scripts around the experiments, such as figure generation, evaluation, and data bookkeeping, which we ran and checked ourselves. No AI-generated text, claim, or number was used without our checking it: each figure is regenerated from its source script, and the labels and values it shows are compared against the text.

The LLM agents studied in this paper are part of the method under evaluation rather than an aid to writing: their roles are specified in §4, and their decisions are what the measurements in §5 quantify. The default controller is a hosted model, and its inference is part of the experimental setup.

We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work trains small language models from scratch on a public corpus. It involves no human subjects, no personal or sensitive data, and no interaction with users: what the system produces is training configurations, not text or judgments that reach people. The concerns it raises are therefore indirect, and two are worth naming. First, automated tuning lowers the cost of running large training jobs without close human supervision. The guardrails and the reversible decision interface of §4 are a partial answer to that risk rather than a complete one, and the mechanism is neutral with respect to how a practitioner chooses to use it. Second, when the controller is answered by a hosted model, every decision sends a description of the training state to an external service, which matters for anyone applying the method to private data; our experiments use public models and public data only. The authors declare no competing interests.

REPRODUCIBILITY STATEMENT

All models and data are public: GPT-2 124M (Radford et al., 2019) and Qwen3-0.6B together with Qwen3-MoE-Tiny (Yang et al., 2025) as backbones, and FineWeb-Edu (Penedo et al., 2024) as the corpus. The training configuration, the per-baseline hyperparameters, and the evaluation protocol are given in Appendix F; the main loop is Algorithm 1; the guardrail rules are Table 4 and Appendix D; the situation description passed to the controller, the experience-store schema, and the retrieval scoring are in Appendices B to E; the analysis of the critical-batch estimate is in Appendix C.

One limit is worth stating plainly. The default controller is a hosted model at temperature 0.3, so the exact sequence of decisions is not reproducible in the strict sense. Everything that shapes a decision is nonetheless fixed and reported: the prompts, the confidence threshold, the guardrails, the starting configuration, and the token budget. Table 5 shows how little the outcome moves when the backend is swapped, so the results can be reproduced in substance even where the exact trajectory cannot.

REFERENCES

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. In *arXiv preprint arXiv:1803.05457*, 2018.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2024. URL `https://zenodo.org/records/10256836`.

Priya Goyal, Piotr Dollár, Ross Girshick, et al. Accurate, large minibatch SGD: Training ImageNet in 1 hour. In *arXiv preprint arXiv:1706.02677*, 2017.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies, 2024.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Iordache, Raia Hadsell, and Koray Kavukcuoglu. Population based training of neural networks, 2017.

Andrej Karpathy. nanoGPT: The simplest, fastest repository for training and finetuning medium-sized GPTs. `https://github.com/karpathy/nanoGPT`, 2022.

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

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. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 2, pp. 230–246, 2020.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Siyi Liu, Chen Gao, and Yong Li. AgentHPO: Large language model agent for hyper-parameter optimization, 2024.

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

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

Sam McCandlish, Jared Kaplan, Dario Amodei, et al. An empirical model of large-batch training. In *arXiv preprint arXiv:1812.06162*, 2018.

Paulius Micikevicius, Sharan Narang, Jonah Alben, et al. Mixed precision training. In *International Conference on Learning Representations (ICLR)*, 2018.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, et al. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.

Anjali Rao and Nikhil Kamalkumar Advani. AI training manager: Bounded closed-loop control of adaptive training recipes, 2026.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, pp. 90–95, 2011.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, et al. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019.

Noah Shinn, Federico Cassano, Ashwin Gopinath, et al. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, et al. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations (ICLR)*, 2018.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text (W-NUT)*, pp. 94–106, 2017.

Thomas Wolf, Lysandre Debut, Victor Sanh, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP: System Demonstrations*, 2020.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

Greg Yang, Edward J. Hu, Igor Babuschkin, et al. Tensor programs V: Tuning large neural networks via zero-shot hyperparameter transfer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Jiayi Zhang, Jinyu Xiang, Zhenrui Yu, et al. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025a.

Wentao Zhang, Yang Young Lu, and Yuntian Deng. Interactive training: Feedback-driven neural network optimization, 2025b. EMNLP 2025 System Demonstrations.

Wentao Zhang, Xuanhe Pan, Han Zhou, Yang Lu, and Yuntian Deng. Interactive training 2: Auditable control plane for live model training, 2026.

A Distributed Runtime and Implementation Details

The main loop. Algorithm 1 gives the control flow of §4 in full; the prose of the body states the same steps in words.

Workload representation. The static configuration fields are the model scale and architecture, the vocabulary and tokenizer, the maximum sequence length, the dataset and token budget, the optimizer, the scheduler shape, the parallelism degree, the precision, and the hardware configuration.

Algorithm 1 EvoForge main loop for online tuning within a single task

Require: Training task T (distributed, with an interaction server), tuning policy LLM, experience store E

Subscribe to the metric stream of T (WebSocket); history $H \leftarrow \emptyset$
while T is not finished **do**
  Append loss / lr / B_{simple} / checkpoint events to H
  if training has passed a pending rollback point **then**
    Dispatch post-rollback hyperparameters (once)
  end if
  if observation window reached and enough history points **then**
    $phase \leftarrow$ classify phase from the relative loss drop δ over the window
    $exp \leftarrow$ retrieve top-k cases and rules from E
    $a \leftarrow \text{LLM}(\text{state}, phase, exp, \text{decision history})$
    if $a =$ `update` **then**
      Three-layer guardrail validation; if passed, apply at the next step boundary
    else if $a =$ `rollback` **then**
      $plan \leftarrow$ rollback agent (checkpoint, post-rollback hyperparameters)
      Dispatch `load_checkpoint`($plan$.target)
      On confirmation: truncate H to the rollback point; stage the post-rollback hyperparameters
    end if
    Record the decision (rationale, confidence, observation window)
  end if
end while
Reflection agent reviews H and the decision log, then writes cases and rules to E

The dynamic trajectory signals are the loss and validation-loss series, the effective learning rate, the gradient norm, the critical-batch estimate, peak memory and throughput, the current batch configuration $(\text{GBS}, \text{MBS}, \text{GAS})$, and the record of past interventions with their outcomes. Retrieval first filters on the static fields and then ranks by similarity over the dynamic series, so that a case drawn from a different model scale cannot enter the ranking at all.

Three coupled quantities. A batch is described by the global batch size GBS, the per-replica micro-batch size MBS, and the gradient accumulation steps $\text{GAS} = \text{GBS}/(\text{MBS} \cdot W)$; the object of an action is therefore a set of mutually constrained settings rather than a scalar. This carries the central tension of the problem: a larger batch gives a more accurate gradient per step but fewer steps under the same budget, so the batch size is neither the larger nor the smaller the better, and the value that suits the phase drifts—which a static configuration cannot track, and a controller that evolves with training can.

Three runtime invariants. Correctness in a distributed setting depends on three invariants. They are not methodological claims, but they determine whether the mechanisms of Sections 3 and 4 work at all.

1. **The same change at the same global step.** Because the step boundary is implicitly synchronized within the data-parallel group by the all-reduce, a control command that takes effect at the boundary guarantees that all replicas read the same configuration. Commands carry the target global step, and each replica checks its own progress before executing, which prevents a configuration split when a single replica lags.
2. **No repeated and no skipped samples across a batch change.** Changing the global batch size or the number of accumulation steps changes how many samples a step consumes; if the batch is changed without adjusting the data loader offset, samples are re-consumed or whole ranges are skipped. Each replica's read offset is recomputed from the number of samples consumed at every batch change.
3. **Step counts recomputed under a conserved sample budget.** Under a fixed token budget, a batch change changes the total number of steps. The budget target is defined over consumed tokens rather than steps, so enlarging the batch mid-run cannot silently increase consumption; this is also what makes the comparison in Section 5 fair.

Asynchronous decisions. LLM inference runs in a background thread and does not occupy the training thread. Only the application of a command requires pausing the loop, and only until a single parameter substitution at the boundary. The measured median pause incurred by an application is 0 ms.

B Tuning Agent Implementation Details

Prompt composition. Blocks are concatenated in this order: role definition; current step with the learning rate and a description of the scheduler; loss curves (recency-biased downsampled); a gradient-noise-scale block; a memory-and-throughput block; a training-phase block holding the phase label and its guidance text; a related-experience block holding the retrieved case cards and rules; several past decision records, including blocked and rolled-back ones with their rationale and confidence; and the current hyperparameters with their adjustable ranges.

Recency-biased downsampling. The loss block takes a fixed number of sample points whose positions are mapped onto the time axis by a power exponent greater than one, packing the points towards the most recent steps: sampling density rises as the present is approached and falls away with distance. A larger exponent gives a stronger bias. The number of points is fixed, so the prompt does not grow as training proceeds; the downsampled series together with the static fields forms the representation snapshot at that moment, updated as training advances.

Decision interface. The LLM is forced to return a fixed-schema JSON object:

```
{
  "action": "update | keep | rollback",
  "updates": {"lr": 1.2e-4, "global_batch_size": 256},
  "reasoning": "rationale of at most 100 characters",
  "confidence": 0.85,
  "min_steps_to_observe": 200
}
```

`min_steps_to_observe` is clamped between a hard lower and upper bound, and `confidence` is the gating input of the admission layer (Appendix D). In runs where gating and validation were active, the model used the middle of the range for the former, with a mean of about 174–242 steps and a spread from 20 to 2000, which indicates that the degree of freedom was exercised rather than collapsed onto a bound.

Phase classification. The window is a fixed number of recent steps. The relative loss drop is $\delta = (\bar{\ell}_{\text{first quarter}} - \bar{\ell}_{\text{last quarter}})/\bar{\ell}_{\text{first quarter}}$, computed on a recency-biased downsampled series so that a single noisy point cannot move the label. A plateau is declared when $|\delta|$ falls below a tolerance, a rise when it is negative beyond the same tolerance, and the early/mid split by how far the run has progressed through the budget; the *final* phase is entered on token progress alone. Each label carries a block of guidance text, and the plateau block additionally carries the instruction to inspect the recent decision history for a self-oscillation pattern.

Predictive memory guardrail. During prompt construction a static estimator (weights, gradients, optimizer state, activations, communication buffers) predicts peak memory and a binary search reports the largest safe micro-batch for the current configuration, which is passed to the model along with the rest of the information so as to steer it towards higher hardware utilization; at validation time any proposal whose predicted headroom is insufficient is rejected.

C Online Critical-Batch Estimation: Derivation and Constraints

The batch signal. The principled basis for a batch decision is the gradient noise scale (McCandlish et al., 2018): the simple noise scale $B_{\text{simple}} = \text{tr}(\Sigma)/\|G\|^2$, formed from the per-sample gradient covariance Σ and the true gradient G, is the batch size at which the marginal benefit of adding data per step begins to saturate. It is not directly observable; the original work supplies an estimator needing only mini-batch gradient norms at two batch sizes. The noise-scale analysis and the batch–learning-rate scaling rules drawn from it are well covered in the literature (Goyal et al., 2017; Smith

Layer	Role	Content
1. Proposal	LLM	Structured JSON: updates, rationale, confidence, observation window; conditioned on phase guidance and recent decisions
2. Validation	Code	Coupled rescaling ($\sqrt{\text{GBS}}$); batch monotonicity; cooldown after a batch-only change; $\text{GAS} \geq 2$; predictive memory guardrail; range truncation. Rejections return an explanation, with at most 3 retries
3. Admission	Gate	Applied only when the self-reported confidence is $\geq$ the threshold; otherwise recorded as `[WITHHELD]` and kept visible in future prompts

Table 4: The three-layer guardrail architecture. The validation layer rejects and explains rather than silently correcting, so that the LLM can repair its own proposals.

et al., 2018; Shallue et al., 2019), so we use the definitions and the estimator without repeating the derivations, and ask only what changes once they enter a closed loop: the coupling is identified in §3 and its handling is in §4.2.

The estimator needs mini-batch gradient norms at two batch sizes, and implementing it directly in distributed training would cost an extra forward or backward pass. We instead obtain **both observations from gradient accumulation**. Suppose each data-parallel replica holds N accumulated micro-batches of size b, for W replicas in total; micro-batch i contributes $d_i = g_i/N$ to `.grad`. We track two scalars, $S_d = \sum_i \|d_i\|^2$, accumulated during the backward pass by a leaf autograd hook, and $B_d = \|\text{grad}\|^2$, read once accumulation completes. Because the path we build on makes `.grad` a MEAN all-reduce, B_d corresponds directly to the large batch NbW, and S_d is averaged across replicas. Substituting into the estimator and solving the linear system gives

$$\|\hat{G}\|^2 = \frac{N(WB_d - S_d)}{NW - 1}, \quad \hat{S} = \frac{NbW(NS_d - B_d)}{NW - 1}, \qquad B_{\text{simple}} = \frac{bW(NS_d - B_d)}{WB_d - S_d}. \tag{4}$$

Since the ratio is extremely noisy at a single step, we follow the original work and apply an exponential moving average with decay β to the numerator and to the denominator *separately* before dividing. It is this separate smoothing that exposes the loop's bias: because the denominator tracks $\|\hat{G}\|^2$, a change of learning rate moves the reported B_{simple} even though the data distribution and the model scale are unchanged, which is why the cooldown of Appendix D is imposed before a batch-only change.

Two further constraints follow from the implementation, and both must hold. The estimator degenerates at $N = 1$, where the denominator loses its degree of freedom, so we require $\text{GAS} = N \geq 2$; this is not a convergence constraint but an *observability* constraint, since at $\text{GAS} = 1$ the estimator returns nothing and the controller loses the batch signal entirely. And the measurement must be taken *before* gradient clipping, because clipping is a nonlinear transform that destroys the linear relation on which the estimator depends.

D Safety and Rollback-Recovery Implementation Details

Validation-layer criteria. The checks are: the $\sqrt{\text{GBS}}$ scaling law with a tolerance of $\pm 25\%$; GBS monotonicity; the cooldown required after a batch-only change; $\text{GAS} \geq 2$; the predictive memory guardrail (Appendix B); and range truncation for every hyperparameter. A rejection returns an explanation and the acceptable range, with at most three retries. The three batch-related criteria are stated in full below.

Coupled rescaling. When a decision changes GBS it must rescale the learning rate within the same decision to $\text{LR}_{\text{new}} = \text{LR}_{\text{old}} \cdot \sqrt{\text{GBS}_{\text{new}}/\text{GBS}_{\text{old}}}$, with a tolerance of $\pm 25\%$; violations are rejected. Two reasons underlie the choice of $\sqrt{\cdot}$. First, training uses fused AdamW, whose update $\text{LR} \cdot \hat{m}/(\sqrt{\hat{v}} + \epsilon)$ cancels the gradient scale between numerator and denominator, so the learning rate directly sets the step size; the linear batch–learning-rate scaling law derives from a signal-to-

noise argument for SGD with momentum (Goyal et al., 2017), and transplanting it to the Adam family turns the entire increase in the learning rate into an increase in step size, which easily causes divergence, whereas $\sqrt{\cdot}$ scaling amplifies far more gently. Second, there is a control-side effect: raising the learning rate together with the batch inflates $\|\hat{G}\|^2$ and cancels the overestimate that a larger batch would otherwise cause.

Cooldown. If a decision changes GBS without touching the learning rate, at least one exponential-moving-average window must have elapsed since the last learning-rate change. With the numerator and the denominator smoothed separately at decay β, the effective window is $1/(1-\beta)$ steps (100 steps at $\beta = 0.99$), the time required for B_{simple} to reflect the current learning rate rather than a mixture with history. When a proposal is blocked, the rejection message reports the remaining steps and offers a coupled alternative.

Batch monotonicity. GBS may not fall below the largest value reached so far. The constraint targets oscillations that repeatedly raise and lower the batch and waste large numbers of steps, and encodes an empirical asymmetry: B_{simple} drifts upward during training, so a batch temporarily above the estimate is a reason to hold rather than to shrink. The high-water mark is reset after a rollback (state-consistent rollback below).

Admission-layer gating. A proposal is applied only when the self-reported confidence is at least the threshold (0.7 by default); otherwise it is recorded as `[WITHHELD]` and kept in the decision history. In runs where gating was active, this layer fired four to five times per run, always on proposals with a confidence of 0.60–0.65.

Rollback trigger and gating. The trigger is judged by the tuning agent, which emits the rollback action together with a symptom description and a suspected starting step; an ordinary plateau or a slight dip is not a case. Gating requires the self-reported confidence to exceed the rollback threshold (0.7 by default) and the number of steps since the previous rollback to exceed the cooldown value.

State-consistent rollback. Reloading a checkpoint moves the training side's global step backwards, so the agent side must retreat in step; otherwise it reads the trajectory of the revoked branch. On confirmation, the training and validation loss curves, the batch estimate curve, and the learning-rate curve are truncated to the restore point; decision records after that point are marked as rolled back, remaining visible but excluded from valid experience during reflection; the batch-estimate snapshots and token progress are invalidated and rebuilt; and the batch high-water mark is reset to the batch size of the restored configuration once training passes the restore point, since otherwise the batch monotonicity criterion above conflicts with the smaller batch that the rollback reintroduces.

Deferred dispatch of post-rollback hyperparameters. The checkpoint-reload success event is emitted *before* the reload completes; dispatching a hyperparameter update at that moment would let the command be overwritten by the optimizer state restored immediately afterwards. Post-rollback hyperparameters are therefore staged rather than dispatched on confirmation, and applied once, after training is observed to have passed the restore point.

Deterministic fallback. When the LLM is unavailable, or the plan it returns is unusable, the system restores the checkpoint from just before the most recent effective change and reverts the hyperparameters involved in that change to their previous values. The fallback exists so that recovery does not depend on the availability of an external model; it is not a substitution of rules for judgment.

E Experience Store Structure and Retrieval

Case cards carry: the situation (training phase, state summary, tags, model and scale), the action (hyperparameters changed and by how much), the outcome (change in loss, whether a rollback was triggered, whether the proposal was blocked), a lesson stated in one sentence for future decisions, and a timestamp with the identifier of the originating task.

Rule items carry: the guidance statement; the support count s and opposition count c; the source (system-induced, entered by hand, or from the literature); the confidence, computed from s and c for induced rules and assigned by the entrant for external ones; and the applicability conditions (phase and model-scale range).

Retrieval proceeds in three steps: filter the candidate set by phase label and model scale, down-weighting cross-scale items; within the candidate set, match the current phase and recent state text by similarity, computed as token overlap and weighted by outcome severity so that failures and rollback cases, as well as successes, rank above neutral ones; and return the most relevant case cards together with the promoted rules applicable to that phase.

Reflection. The reflection agent runs once per completed task. It reads the full decision log and the metric trajectory, classifies every recorded decision as kept, applied, rolled back, or withheld, and writes out: cases worth reusing—those that clearly helped, those that clearly hurt such as a decision that led to a rollback, and those exposing a recurrent trap—and rules that the evidence of this task supports or contradicts. Cases are appended; rules are matched against existing items, merged when semantically equivalent, and their counts accumulated. If the LLM is unavailable, the agent falls back to one coarse-grained case built from the decision log, so that a completed task never leaves the store unchanged.

Promotion and deduplication. Rules produced by the reflection agent are compared against existing items: semantically equivalent ones are merged and their counts accumulated, and new ones enter with their initial counts. Only rules whose support reaches the lower bound and whose confidence exceeds the threshold join the set injected into prompts; those that do not are kept in the store and continue to accumulate evidence. Confidence is updated with the counts, so a rising opposition count lowers the confidence of a rule already promoted; once it falls below the threshold the rule leaves the injection set and returns to accumulating evidence. Rules entered by hand or drawn from the literature skip this procedure and take part in injection at the confidence assigned by the entrant.

F Experimental Setup Details

Model and optimizer. GPT-2 124M has 12 layers, 12 heads, $d = 768$, and a vocabulary of 50,257. Training uses fused AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$) (Loshchilov & Hutter, 2019) in bfloat16 mixed precision (Micikevicius et al., 2018).

Training setup. Every run trains for one epoch on FineWeb-Edu (Penedo et al., 2024) over 8-way data parallelism, with the validation loss computed on a held-out split every 20 steps. The budget is 10B tokens counted over consumed tokens rather than steps (Appendix A), so that a batch change made mid-run cannot alter it silently.

Baselines. All three start and end at $\mathrm{GBS} = 512$, $\mathrm{MBS} = 32$ and complete 18,949 updates; sharing configuration and budget, they differ only in schedule shape and implementation and form an internal control. *cosine*: peak learning rate 6×10^{-4}, 1% warmup, weight decay 0.1; the baseline was tuned rather than set casually, sustains about 672k tokens/s, and finishes the budget in 4.13 hours. *WSD*: a warmup–stable–decay three-phase schedule with the same peak rate, warmup, and weight decay, and a 10% decay phase (Hu et al., 2024). *nanoGPT*: the same budget under the nanoGPT reference implementation (Karpathy, 2022).

EvoForge. The system starts from $\mathrm{GBS} = 64$, $\mathrm{MBS} = 4$, eight times smaller than the baselines, and raises the global batch during training, under the controller setting of §5.1.

Metrics and protocols. We report final validation loss, and the numbers of interventions, guardrail rejections, and rollbacks. Five downstream tasks are evaluated with the language-model evaluation harness (Gao et al., 2024): ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), and COPA (Roemmele et al., 2011). The "Avg." column of Tables 1 and 2 is the mean over the five.

G Additional Result Figures

This appendix complements §5.2 with two further figures and the reading they support.

Figure 4 places the cooldown cuts of Figure 3 in the full decision trajectory of the run, against the static baseline on a common token axis: the global batch rises from 64 to 256 through gradient accumulation and to 384 near the end, while the baseline stays at 512; the weight decay goes from 0.1 to 0, and the second Adam moment from 0.95 to 0.99 and then 0.999; a learning-rate probe at

6.7B costs 0.028 in loss and is reverted, and that probe marks the start of the late phase, within which the cooldown cuts fall.

Figure 5 repeats the comparison on the downstream tasks every billion tokens. EvoForge is already ahead on all five at 1B (+2.2, +1.5, +1.6, +3.8, +8.0 points), holds the lead on ARC-Easy and HellaSwag at every checkpoint, loses it on PIQA only at 3B, and changes sign repeatedly on SciQ; by 9B all five are ahead and the five-task mean is 54.65 against 52.85. The advantage is thus in place long before the late phase and continues to widen as the budget is consumed.

H Effect of the Controller LLM

A method tied to a single controller would be impractical, so we hold the starting configuration, token budget, and confidence threshold fixed and swap only the controller backend. Each row is an independent run; the default-backend row repeats the configuration of §5.1 and lands at 3.12396 rather than at the 3.1180 of Table 1, so the 0.19% between the two bounds the spread of a rerun. Against that repeat, GLM-5.2 reaches 3.12675 and Qwen3.5-397B-A17B 3.12768, a rise of 0.09% and 0.12%: smaller than the spread between two runs of one configuration, and an order of magnitude below the gap to the static baselines (§5.2).

Guardrail rejections stay at 0–1 and the update count at 12, 10, and 10, so a weaker backend is not merely a more cautious one, and token consumption falls from 1.10M to 0.58M for at most 0.12% in loss.

Controller backend	**Val. loss ↓**	**Δ vs. default**	**Rejected**	**Keep / Update**	**Tokens**
DeepSeek-Flash (default)	3.12396	—	1	144 / 12	1,103,627
GLM-5.2	3.12675	+0.09%	1	99 / 10	775,329
Qwen3.5-397B-A17B	3.12768	+0.12%	0	67 / 10	575,796

Table 5: Comparison of controller backends under the same starting configuration, token budget, and confidence threshold. "Δ" is the relative increase in validation loss over the default backend; "Rejected" counts guardrail-rejected proposals; "Keep / Update" counts the two decision types emitted by the controller; "Tokens" is the total number of prompt plus completion tokens consumed. Each row is an independent run.

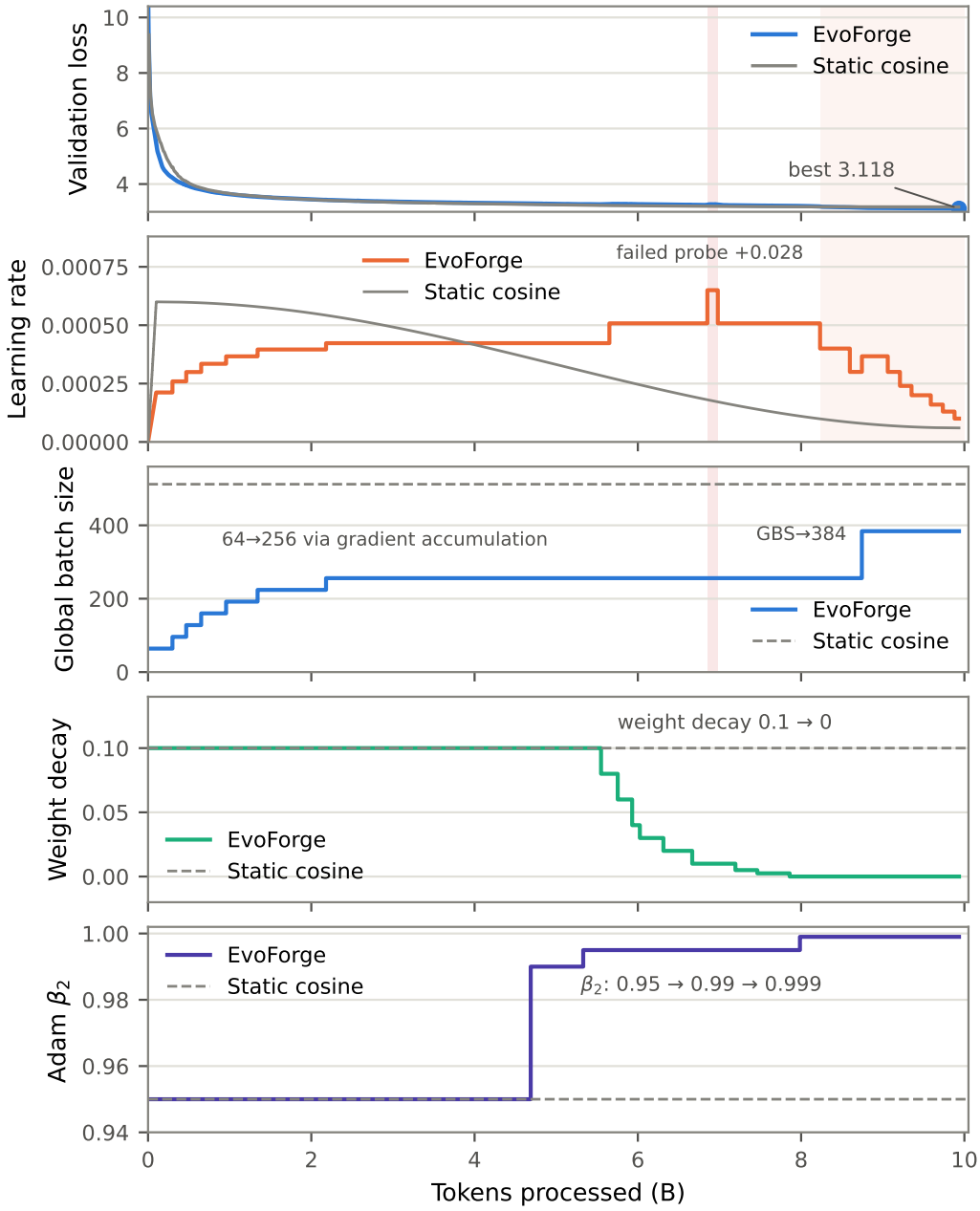

Figure 4: The decision trajectory of the 124M run against the static baseline, on a common token axis. From top to bottom: validation loss, learning rate, global batch size, weight decay, and the second Adam moment. The shaded region marks the late phase, entered after the learning-rate probe at 6.7B.

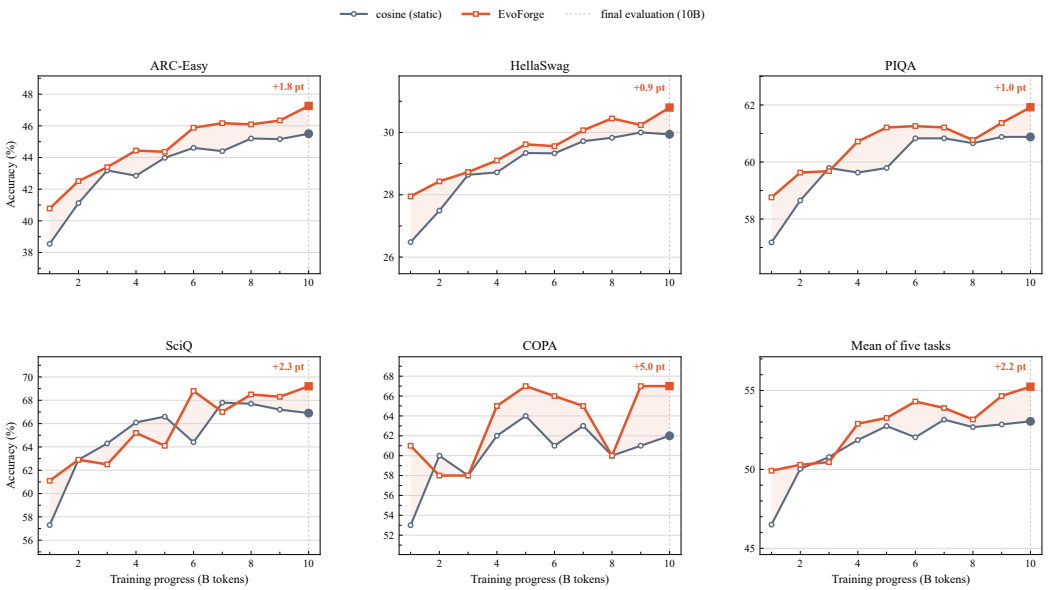

Figure 5: Downstream zero-shot accuracy against training progress on the 124M run, evaluated every billion tokens. The baseline is the tuned cosine schedule of Table 1; both curves are single-seed.