



the intelligence layer for AI training

Paramorph

Technical report

Learned control of training hyperparameters
with multi-agent reinforcement learning

Paramorph: technical report

Learned, adaptive control of neural-network training with multi-agent reinforcement learning

Recurvia Ltd, London

Authors Recurvia team

Date October 2026

1 Executive summary

Paramorph is a foundation model for training control. It watches a neural network as it trains and, every hundred batches, resets the learning rate, Adam $\beta 1$ and $\beta 2$, dropout and weight decay according to what the run is actually doing. It is trained with multi-agent reinforcement learning across a large population of training runs, and it is used zero-shot: a new training run gets an adaptive, run-specific schedule without a hyperparameter search.

This report presents the internal evaluation of the latest large-scale training run of the Paramorph foundation policy. Every result comes from inner tasks that Paramorph did not see during training, and many of them lie outside its training distribution altogether: larger models, larger batches and an entirely unseen model family. The main findings are summarised below.

67–184

trials of a 256-trial, five-hyperparameter search needed to match a single stable zero-shot Paramorph run

1.3–2.2×

faster than a 256-trial learning-rate search at every GPT-2 size evaluated, from 5M parameters to the official 124M model

2.8–23×

median speed-ups against five untuned baselines, from constant settings to the scheduled learning rate of the Hugging Face Trainer, across the 41-task suite

- **One zero-shot run replaces a large search.** On `gpt2-nano`, a 256-trial search over the learning rate and its schedule never reached the loss Paramorph achieved in one run, and a 256-trial search over all five controlled hyperparameters needed about 67 trials to match Paramorph's weakest seed and about 164 to pass all five. On the official 124M-parameter GPT-2, the full search needed about 184 trials to pass the weaker of the two stable Paramorph runs, and its final best only matched the stronger one.
- **Stable scaling to 1B parameters on an unseen family.** On OLMo 3 models from 14M to 1B parameters, a family absent from Paramorph's training, every seed trains stably at every size and Paramorph improves on its own starting configuration throughout.
- **Generalisation beyond the training distribution.** Speed-ups hold on unseen architectures (OLMo 2 and OLMo 3), on models larger than any of their family seen in training (a Qwen3 model, the official 124M GPT-2 and OLMo 3 up to 1B) and on batch sizes up to four times larger than any seen in training. They also hold when the training budget is varied: most results in this report use a budget of 256 Paramorph decisions (about 25,600 batches), and the speed-up rises from about 2× at short budgets to 24.8× at 4,096 decisions (about 410,000 batches), sixteen times the standard length.
- **Consistent gains over standard defaults.** Across the 41-task suite, Paramorph matched or beat the baseline within the training budget in 95% of all evaluation runs (five seeds per task, against the baselines run for each task). The median speed-up is 2.8× over PyTorch defaults, 4.0× over Paramorph's own fixed starting configuration and 5.4× over the same configuration with its learning rate rescaled for each task. All three references are constant configurations, so we treat them as

measures of what a practitioner gains on a first, untuned run, and rely on the tuned comparisons above for the strength of the method.

- **Interpretable, adaptive behaviour.** Paramorph is not a language model post-trained with reinforcement learning, and it has no pre-trained knowledge of machine-learning practice to draw on. It is trained from scratch with classical reinforcement learning, purely from the experience of controlling training runs. Without being told to, the policy discovers warm-up and decay, switches off dropout for single-pass pre-training, anneals weight decay and selects a much lower peak learning rate for a larger model. It produces different schedules for different seeds of the same task, reacting to each run as it unfolds.
- **A growing record of training runs.** Every training and evaluation run is logged as a trajectory of observations, decisions and outcomes. Whether these can be used to improve the policy through offline reinforcement learning is an open research question, on which Recurvia has published foundational work at the Reinforcement Learning Conference in 2026 (Section 3.4).

WHAT THE EVIDENCE SHOWS, AND WHAT COMES NEXT

The evaluation also shows clearly where the current policy is weakest. Some seeds become unstable on the official 124M GPT-2 and at the largest batch sizes, the speed-up ratio over constant configurations narrows as models grow (although the absolute time saved per run does not, Section 7.7), and the policy does not yet fully exploit the training budget it is given. Two things contribute. These regimes were thin in Paramorph's training distribution, because large inner runs are expensive at the scale reinforcement learning requires. Widening that coverage is one lever. A second is already in hand: a review of the training stack behind this run has identified specific, testable improvements to how the policy is rewarded, what it observes and how it is optimised, which target stability and budget use directly. None of them was applied in the run reported here, so these results are a starting point rather than a ceiling. Validating each improvement at scale and against strong baselines takes many controlled experiments. That programme is under way, supported by compute from the European AI Factories programme, through which Recurvia has received 200,000 GPU hours. In parallel, a policy trained on a specific, recurring workload is expected to perform best on that workload, and bespoke policies of this kind are a second direction we are pursuing alongside the general foundation policy (Section 8).

HOW TO READ THE NUMBERS

A **speed-up** of $3\times$ means the baseline needed three times as much training data as Paramorph to reach the baseline's own best validation loss within a fixed budget. It is a measure of data and compute saved at equal quality. Detailed results use five independent seeds per task, all with a single Paramorph checkpoint from late in training, and all illustrative tasks were chosen before results were compiled (Section 4). The raw evaluation runs behind every figure are publicly viewable in Metrana (Section 4.5).

- 2 The problem: hyperparameters are expensive to tune and static by design
- 3 How Paramorph works
- 4 Evaluation methodology
- 5 Results across the evaluation suite
- 6 Paramorph against tuned baselines

7 Generalisation and reliability

8 Current limits and next steps

9 Conclusion

A Appendix: definitions and experimental protocol

B Appendix: research progress, September 2025 to September 2026

C Appendix: per-task results

2 The problem: hyperparameters are expensive to tune and static by design

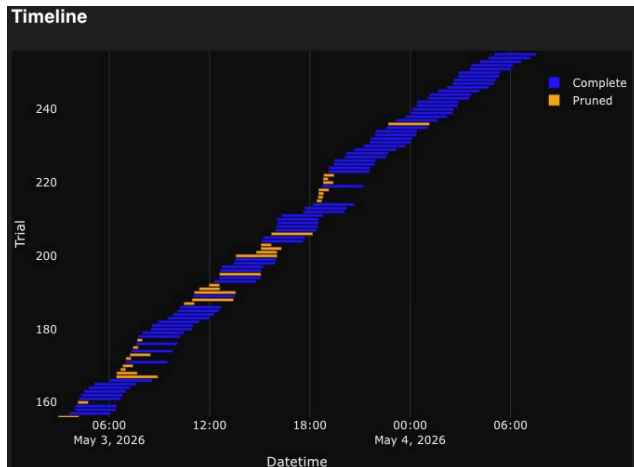
The training hyperparameters of a neural network, such as the learning rate, the Adam momentum terms, dropout and weight decay, have a large effect on how quickly a model learns and how good it becomes. Practitioners normally choose them by running a hyperparameter search: many training runs with different settings, keeping whichever performs best. This approach has three structural costs.

First, search is expensive. The number of candidate configurations grows combinatorially with the number of hyperparameters tuned jointly, and it grows again once schedules are involved, because the shape of a learning-rate schedule has its own parameters (warm-up length, decay shape, final value). Each candidate is a training run. Figure 1 shows part of a single tuning study from this evaluation: the last hundred trials alone span more than a day of wall-clock time, and the per-trial loss curves show how much of that compute is spent on configurations that are discarded.

Second, the result is static. In standard practice most hyperparameters, including the Adam momentum terms, dropout and weight decay, are held constant for the whole run. The learning rate is usually the only one that changes, and it does so along a predetermined schedule, such as a linear warm-up followed by cosine decay, whose shape is set by further hyperparameters (warm-up length, peak value, decay rate, final value) that are themselves fixed in advance. Even a well-tuned schedule is therefore fixed before the run begins. It is a prediction of how training will go rather than a response to how it does go, so it cannot react to a loss spike, a plateau or a change in gradient statistics. A momentum setting that suits the early, fast-moving phase of training is rarely ideal once the loss has flattened.

Third, tuning does not transfer. When the model size, dataset, batch size or training length changes, the optimal configuration moves and the search must be repeated. Organisations that train many models pay this cost repeatedly.

Trial timeline (trials 157 to 255)



Validation loss of each trial during training

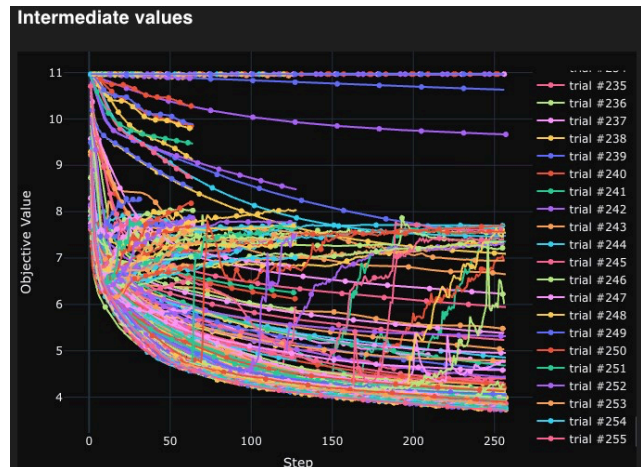
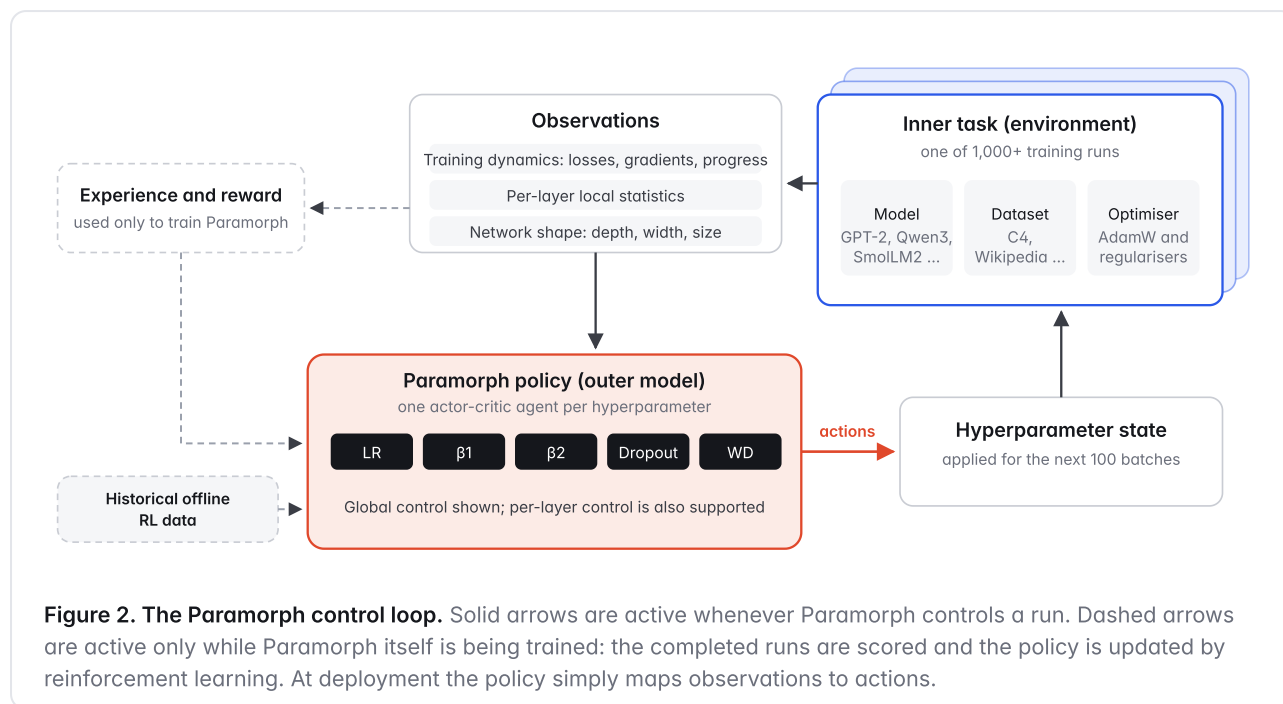


Figure 1. The cost of conventional tuning. Excerpts from the 256-trial Optuna study used to build the fully tuned baseline for `gpt2-off_c4`. Left: trial start and end times; blue trials ran to completion and orange trials were pruned early. Right: validation loss against training progress for each trial. The best trial reached a validation loss of 3.715.

Paramorph is designed to remove all three costs at once. It replaces the search with a learned policy that is applied zero-shot, it adapts every controlled hyperparameter continuously during the run, and because it is trained across a broad distribution of models and settings, it carries what it has learned to new workloads without retuning.

3 How Paramorph works

Paramorph treats a training run as a reinforcement-learning environment. The run being optimised, the **inner task**, consists of a model, a dataset, an optimiser and a training configuration. Paramorph itself, the **outer model**, is the agent. At a fixed cadence, every 100 batches in the experiments reported here, Paramorph reads a compact summary of the state of the run and outputs new values for each hyperparameter it controls. Training then continues with those values until the next decision. Acting at this coarser cadence keeps the control problem tractable while still letting the policy track the trajectory of the run closely.



3.1 Observations, actions and reward

Paramorph never sees a user's data or model weights. Its observations are summary statistics that are cheap to compute and independent of the training framework: training and validation losses, gradient statistics, progress through the training budget, per-layer local statistics and descriptors of the network's shape. Shape descriptors are what allow a single policy to generalise across architectures rather than memorising one.

The action space in the run evaluated here is the global value of five hyperparameters: the learning rate, Adam β_1 and β_2 , the dropout rate and weight decay. The system already supports additional hyperparameters and per-layer control, for example a separate learning rate for every layer of the inner model. The run evaluated in this report uses global control. In earlier experiments, per-layer control reached better final performance than global control on small models, but on larger models it trained less stably. Because per-layer control can always reproduce a global schedule, this gap reflects multi-agent training details still to be resolved rather than a limit of the approach, and per-layer control remains on the research roadmap (Section 8 and Appendix B). The reward reflects how well the inner training is progressing. The exact design of the observations, actions and reward is proprietary and, as is typical in applied reinforcement learning, getting it right has been a substantial part of the research programme.

3.2 Multi-agent training

Each controlled hyperparameter is assigned its own agent, an actor-critic pair. The actor sets that hyperparameter at each decision step and the critic estimates the value of the current state. All agents share the same observations and the same reward, so they learn to coordinate rather than compete; the interacting learning-rate and momentum schedules in Section 5.3 are consistent with this design. Paramorph is trained from scratch with classical reinforcement learning. Unlike agentic approaches, which post-train a pre-trained language model with reinforcement learning, it starts with no prior knowledge of how networks should be trained: everything it does, including the behaviours in Section 5.3, is learned from experience. Each training episode is a complete inner training run, and Paramorph is trained across a broad, deliberately varied distribution of inner tasks so that the resulting policy transfers to new ones.

The programme has focused on transformer language-model training, where compute budgets are largest. Earlier versions of Paramorph were trained on convolutional networks and multi-layer perceptrons and also performed well, so the approach is not specific to transformers. Training and evaluation are run and analysed in **Metra**, Recurvia's own experiment-tracking platform, which we built to observe agent behaviour and environment dynamics in large-scale reinforcement learning; the dark-background charts in this report are taken from it.

3.3 Relation to existing approaches

Choosing training hyperparameters is a long-standing problem, and several families of methods address it. Paramorph differs from each of them in what it learns, when it acts and how much compute it needs per workload. In several cases it is complementary rather than competing.

Multi-run hyperparameter optimisation. The most widely used approaches treat each training run as one evaluation of a black-box function and search over configurations. Grid search tries every combination of a few values per hyperparameter, and its cost grows exponentially with the number of hyperparameters. Random search (Bergstra and Bengio, 2012) samples configurations at random, which is usually more efficient because only a few hyperparameters tend to matter for a given problem. Bayesian optimisation, including the tree-structured Parzen estimator used by Optuna to build the tuned baselines in this report, fits a model of performance to past trials and uses it to choose the next one. Multi-fidelity methods such as Hyperband and ASHA cut costs further by stopping unpromising trials early. Population Based Training (Jaderberg et al., 2017) runs many trials concurrently and periodically copies the weights and hyperparameters of the best performers into the worst, perturbing them as it does so. It is one of the few established methods that produces hyperparameter schedules rather than constants.

All of these are multi-run methods: their cost is measured in tens to hundreds of training runs per workload, and the search must be repeated when the workload changes. Except for Population Based Training, they also return fixed values or fixed schedules. They remain the right tool for *architectural* hyperparameters, such as the number of layers, the hidden dimension and the number of attention heads, which define the model and cannot sensibly change while it trains. Paramorph is designed to work alongside them. Within each trial of an architecture search, Paramorph controls the training hyperparameters, so the outer search covers far fewer dimensions and every candidate architecture is trained under an adaptive schedule rather than a guessed one.

The cost of this approach in production is something members of the Recurvia team know at first hand. At Apple, one of Recurvia's founders worked on the Siri team, where models were tuned with Bayesian hyperparameter optimisation, and was at times the one running these sweeps. Each supported language needed its own sweep of hundreds of training runs, and the sweeps had to be repeated every few weeks because each new data release invalidated the previous results. Watching that compute spent again and again to rediscover settings that had only shifted slightly motivated the idea that hyperparameter control should be learned once and reused, and that idea led to Paramorph.

Hyperparameter transfer and schedule-robust training. μ P (Yang et al., 2022) and fitted scaling laws aim to tune once on a small proxy model and transfer the result to the target scale. They still require sweeps at small scale and yield static values, and the μ Transfer authors note that regularisation hyperparameters such as dropout and weight decay do not, in general, transfer. Warmup-stable-decay schedules and schedule-free optimisers reduce dependence on knowing the training horizon in advance, but their own hyperparameters still have to be chosen and they do not respond to the state of the run. Paramorph can in principle operate on a μ P-parametrised model or alongside a schedule-free optimiser.

Gradient-based hyperparameter optimisation. A separate line of work treats continuous hyperparameters as variables to be optimised by gradient descent. Maclaurin et al. (2015) computed exact hypergradients by differentiating through an entire training run, which is precise but expensive in memory and difficult to scale. Hypergradient descent (Baydin et al., 2018) adapts the learning rate online using the gradient of the current loss with respect to it, which is cheap but greedy. Implicit-differentiation methods (for example Lorraine et al., 2020) scale to many hyperparameters but rely on approximations to the inverse Hessian. Across these methods, the hypergradient is either expensive to compute exactly or rests on short-horizon or second-order approximations that make it noisy and biased towards short-term gains (Wu et al., 2018). They also apply only to differentiable, continuous hyperparameters. Paramorph optimises the outcome over the whole training budget through reinforcement learning, needs no hypergradients, and handles any hyperparameter it can observe the effect of.

Learned hyperparameter control. The closest prior work to Paramorph is LHOpt (Almeida et al., 2021), from OpenAI, which also uses reinforcement learning to learn a policy that adjusts the hyperparameters of an existing optimiser rather than the weights themselves. It was trained on a distribution of tasks and reported speed-ups of around 2× on ImageNet and 2.5× on language modelling. Paramorph builds on the same framing and differs in its design: a separate agent for each controlled hyperparameter, trained jointly with multi-agent reinforcement learning, a training distribution built from many model families with explicitly varied shapes, and evaluation against tuned baselines on held-out and out-of-distribution tasks.

Learned optimisers. VeLO (Metz et al., 2022), from Google, and its more compute-efficient successor Celo (Moudgil et al., 2025) are also learned, but they learn something different. They are drop-in replacements for Adam: a neural network computes the update for the model's weights directly, so it must act at every training step and make predictions for every parameter, a very large number of decisions per run. VeLO was meta-trained with around 4,000 TPU-months of compute, yet an independent evaluation (Rezk et al., 2023) found that it still needed problem-specific tuning and did not consistently outperform tuned standard optimisers. Celo reduces meta-training cost dramatically. In our own tests it became unstable at training horizons only slightly longer than those reported in its paper, a limitation its authors confirmed to us. Paramorph sits one level above the optimiser instead. It wraps around Adam, or whichever optimiser the user already trusts, as an outer control loop, acting once every 100 batches on a handful of its hyperparameters. It therefore makes several orders of magnitude fewer decisions per run, adds negligible overhead, and leaves the optimiser's well-understood behaviour in place.

Approach	Runs per workload	Adapts during a run	Acts on	Relationship to Paramorph
Grid, random and Bayesian search	Tens to hundreds	No	Any hyperparameter, fixed values	Complementary: best for architectural choices; Paramorph trains each trial
Population Based Training	A population run concurrently	Yes, at population level	Training hyperparameters	Paramorph adapts within a single run, using what it learned across past runs
μ P and scaling laws	Sweeps at small scale	No	Mainly learning rate and initialisation	Complementary: can provide Paramorph's starting point
Gradient-based HPO	One, with extra computation	Yes, greedily	Differentiable hyperparameters	Paramorph optimises the full horizon without hypergradients
Learned hyperparameter control (LHOpt)	One	Yes, periodically	Hyperparameters of the existing optimiser	Closest prior work; Paramorph is multi-agent, with one agent per hyperparameter
Learned optimisers (VeLO, Celo)	One	Yes, every step	Every weight update	Drop-in replacements for Adam; Paramorph wraps around Adam and acts every 100 batches
Paramorph	One	Yes, every 100 batches	Training hyperparameters of the existing optimiser	

This report compares Paramorph with tuned searches and library defaults. Direct comparisons with, and combinations with, μ P and schedule-free optimisers are planned for the next evaluation round.

3.4 A growing dataset of training trajectories

Every training and evaluation run Paramorph takes part in produces a complete trajectory: what it observed, what it decided and how the run responded. Recurvia stores these trajectories as a proprietary dataset, which grows with every experiment and, in future, with every deployment. A dataset of this kind could in future support offline reinforcement learning, in which a policy is improved from previously collected experience without running new training jobs, potentially at a fraction of the compute cost of online training. Whether that is achievable for Paramorph is still an open research question.

Offline reinforcement learning is challenging, and more so in the multi-agent setting Paramorph uses. Each agent's best behaviour depends on what the other agents do, and in a fixed dataset that behaviour cannot adapt as the agents learn, so standard offline methods can settle on poorly coordinated joint policies. Recurvia has been carrying out foundational research on this problem. The resulting paper, *CODA: Coordination via On-Policy Diffusion for Multi-Agent Offline Reinforcement Learning* (Hedman et al., 2026), was published in the Reinforcement Learning Journal and presented at the Reinforcement Learning Conference (RLC) 2026. It uses diffusion models, conditioned on the agents' current joint policy,

to generate synthetic experience that reflects how the agents are changing, recovering the co-adaptation that offline training otherwise lacks. The work was led by Recurvia researchers with collaborators at the Universities of Oxford, Edinburgh and Cape Town and the Politecnico di Milano.

REFERENCES FOR SECTION 3

- Almeida, D., Winter, C., Tang, J. and Zaremba, W. (2021). A generalizable approach to learning optimizers. arXiv:2106.00958.
- Baydin, A. G. et al. (2018). Online learning rate adaptation with hypergradient descent. *ICLR*.
- Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *JMLR* 13.
- Hedman, M., Tessera, K. A., Formanek, J. C., Sims, A., Zamboni, R., McInroe, T., Torr, J. and Fosong, E. (2026). CODA: Coordination via on-policy diffusion for multi-agent offline reinforcement learning. *Reinforcement Learning Journal* (RLC 2026).
- Jaderberg, M. et al. (2017). Population based training of neural networks. arXiv:1711.09846.
- Lorraine, J., Vicol, P. and Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. *AISTATS*.
- Maclaurin, D., Duvenaud, D. and Adams, R. P. (2015). Gradient-based hyperparameter optimization through reversible learning. *ICML*.
- Metz, L. et al. (2022). VeLO: Training versatile learned optimizers by scaling up. arXiv:2211.09760.
- Moudgil, A., Knyazev, B., Lajoie, G. and Belilovsky, E. (2025). Celo: Training versatile learned optimizers on a compute diet. arXiv:2501.12670.
- Rezk, F. et al. (2023). Is scaling learned optimizers worth it? Evaluating the value of VeLO's 4000 TPU months. *NeurIPS Workshop on I Can't Believe It's Not Better*.
- Wu, Y., Ren, M., Liao, R. and Grosse, R. (2018). Understanding short-horizon bias in stochastic meta-optimization. *ICLR*.
- Yang, G. et al. (2022). Tensor Programs V: Tuning large neural networks via zero-shot hyperparameter transfer. arXiv:2203.03466.

4 Evaluation methodology

Because Paramorph is trained across a changing distribution of inner tasks, the usual reinforcement-learning training metrics, such as episode return, are hard to interpret: the return depends on which tasks happen to be sampled. We therefore evaluate Paramorph separately, on a fixed suite of **41 inner tasks**, by training each task for its full budget under the schedule Paramorph produces from a given checkpoint.

4.1 Held-out and out-of-distribution tasks

None of the evaluation tasks appears in Paramorph's training distribution. They fall into two groups. Some are **held out but in distribution**: their configuration (model family, size, batch size, context length) is of a kind Paramorph could have encountered in training. Others are **fully out of distribution**: either they could not have appeared in training at all, because the model family was never included or the batch size exceeds 512, or the model is larger than any that occurred in training in practice. All OLMo tasks fall into the second group.

For reference, the run evaluated here was trained on a roughly equal mix of inner tasks from three model families: GPT-2, Qwen3 and SmoLM2, all trained with the Hugging Face Trainer. There is no explicit upper limit on model size or training length in the training distribution, but in practice almost all training tasks have fewer than 10M parameters (the largest observed has 65M), almost all train for fewer than 1,000 decision steps, and no batch size exceeds 512. Charts in Section 7 mark these limits.

4.2 Baselines

Every result compares Paramorph with a baseline on the same model, data and budget. We use seven baselines, ranging from library defaults to per-task hyperparameter searches.

Baseline	Description	Applies to
FIXED CONFIGURATIONS, NO TUNING COMPUTE		
<code>RecurviaDefaults</code>	Paramorph's own initial hyperparameter values, held constant for the whole run with no warm-up or decay (Paramorph takes no actions). This isolates the contribution of Paramorph's decisions, and is our primary reference.	All tasks
<code>RecurviaDefaultsScaled</code>	As above, but with Paramorph's first action applied: the learning rate is rescaled once for the task's batch size, context length and model size, and all values are then held constant.	All tasks
<code>TorchDefaults</code>	PyTorch default values where defined (e.g. learning rate and Adam β values in <code>torch=2.11.0</code>), otherwise <code>RecurviaDefaults</code> .	All tasks
<code>HuggingFaceDefaults</code>	The learning-rate schedule and constant hyperparameters of the Hugging Face Trainer in <code>transformers=5.5.4</code> .	All tasks
<code>HuggingFaceDefaultsConstantLR</code>	As above, with the learning rate held at its initial value.	All tasks
PER-TASK BAYESIAN HYPERPARAMETER SEARCHES (TPE), 256 TRIALS EACH		
<code>TunedLR</code>	Learning rate and schedule tuned separately for each task; other hyperparameters fixed.	Selected tasks
<code>TunedFull</code>	All five hyperparameters that Paramorph controls tuned jointly for each task.	Selected tasks

Tuned baselines use Optuna's tree-structured Parzen estimator (TPE), a form of Bayesian optimisation, with 256 trials per task. For Hugging Face and OLMo models the learning-rate search space includes a constant rate and cosine and linear warm-up/decay schedules with tuneable parameters; other hyperparameters are searched over constant values. The search minimises final validation loss, and unpromising trials are pruned early.

The two groups of baselines answer different questions. The fixed configurations, like Paramorph, require no tuning compute, so they are a like-for-like comparison of what a practitioner gets on the first run. The tuned baselines are the result of up to 256 training runs per task. Beating, or even matching, them with a single zero-shot run is a large compute saving, and Section 6 quantifies how much.

4.3 The speed-up metric

Our principal measure is **speed-up**. For a given task and baseline, we find the best validation loss the baseline achieves within its training budget and the number of steps it took to reach it. We then find the first step at which Paramorph matches or improves on that loss. The speed-up is the ratio of the two. Figure 3 works through an example.

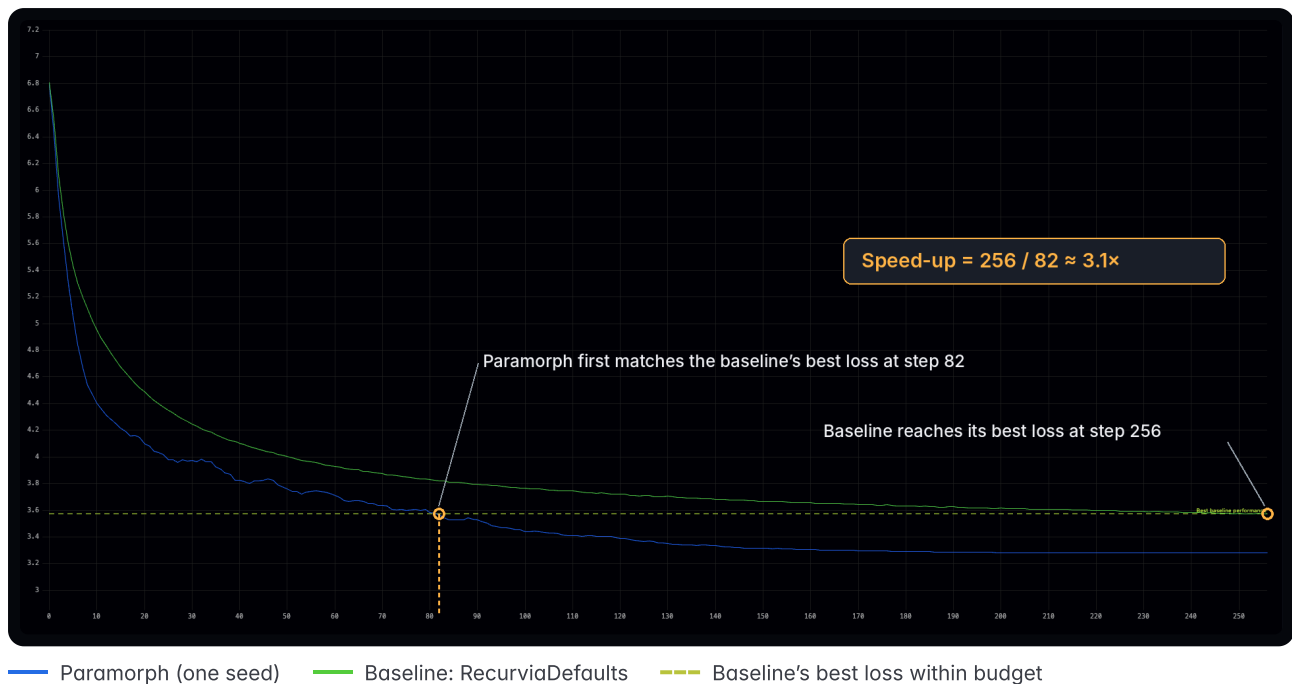
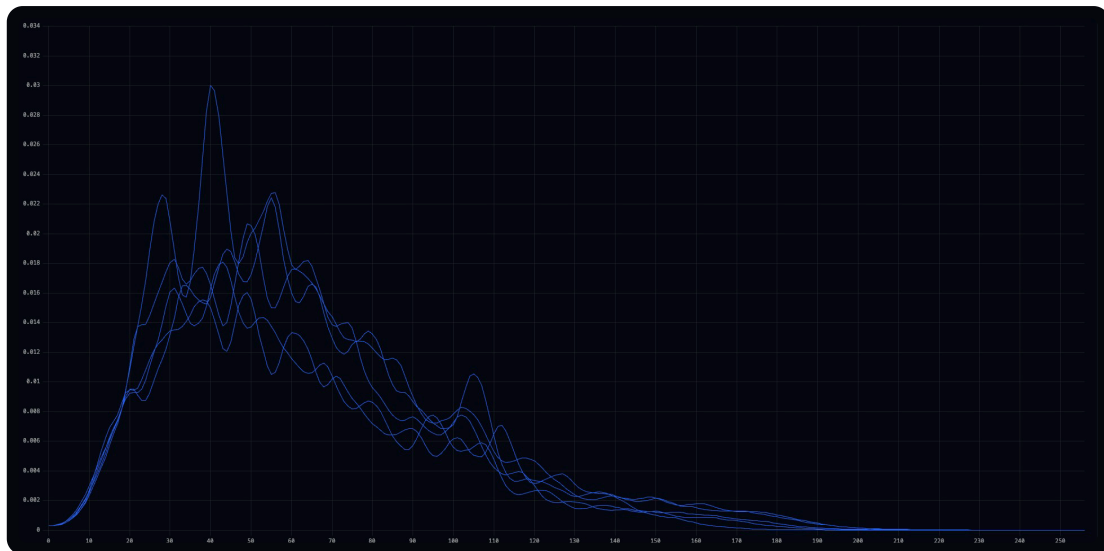


Figure 3. How speed-up is measured. Validation loss on `gpt2-nano-global_wikipedia`, one seed, from an earlier Paramorph checkpoint (episode 200). Horizontal axis: decision steps of 100 batches. The baseline reaches its best loss at the end of its 256-step budget; Paramorph first reaches the same loss at step 82, a speed-up of $3.1\times$. Paramorph then keeps improving and finishes well below the baseline.

Three properties of the metric are worth stating. First, both sides are measured at the earliest step that reaches the target: if the baseline diverges after reaching its best loss at step 100 of 200, its step count is 100, not 200, so divergence reduces the speed-up rather than inflating it. Second, the metric never extrapolates: if Paramorph does not reach the baseline's best loss within the same budget, no speed-up is recorded. Third, because Paramorph usually continues to improve after matching the baseline, the speed-up does not capture its final-quality advantage. Conversely, when a baseline plateaus and improves only marginally late in its budget, the ratio can become large, which is why we summarise results with medians. Speed-up measures how many fewer training steps, and hence how much less data and compute, Paramorph needs to reach the baseline's quality.

4.4 Seeds, checkpoints and selection

Training is stochastic, so a single run is not enough to judge reliability, and the detailed results in this report use **five seeds** per task. Evaluating every Paramorph checkpoint on every seed would be prohibitively expensive, so the two are handled separately: while Paramorph trains, successive checkpoints are evaluated on a single seed to track progress (Section 5.2), and the multi-seed evaluations use one checkpoint taken from late in the training run. Because the policy reacts to what it observes, different seeds of the same task produce different schedules (Figure 4), and we report the mean and standard deviation across seeds. The tasks shown in detail in this report were selected in advance of compiling results, to illustrate each axis of generalisation rather than to show the most favourable cases.



— Learning rate chosen by Paramorph, five seeds of the same task

Figure 4. One policy, five run-specific schedules. Learning rate over 256 decision steps on `gpt2-nano-global_wikipedia`. All five seeds share a warm-up and decay structure, but the peak value, its timing and the fine structure differ because Paramorph responds to each run’s own dynamics.

4.5 Data availability

The raw results behind this report are publicly viewable in Metrana, Recurvia’s experiment-tracking platform, where each evaluation run can be inspected in full:

- All runs from the training run evaluated here:
https://app.metrana.ai/user-06ab261e4005df5af43c73e1ddb027e0/public-paramorph-results/run_2026-09-13/-/runs
- Single-seed evaluations of successive checkpoints during training (Section 5.2):
https://app.metrana.ai/user-06ab261e4005df5af43c73e1ddb027e0/public-paramorph-results/run_2026-09-13/-/run/training-eval/charts
- Five-seed evaluations of the full 41-task suite (Sections 5.1 and 7):
https://app.metrana.ai/user-06ab261e4005df5af43c73e1ddb027e0/public-paramorph-results/run_2026-09-13/-/run/multiseed-eval/charts

Per-task speed-ups against every baseline are listed in Appendix C.

5 Results across the evaluation suite

5.1 Summary over all 41 tasks

Table 1 summarises the full evaluation suite. Against the five fixed configurations, which cost nothing to obtain, Paramorph matches or beats the baseline on every task but one, the shortest training budget against the PyTorch defaults, and is faster on all but two: on the official 124M GPT-2 it only matches the PyTorch defaults (1.0×). Against Paramorph’s own fixed starting configuration the median speed-up is 4.0×, and about a third of tasks exceed 7×; against the same configuration with its learning rate rescaled for each task it is 5.4×. The gains therefore come from the decisions the policy makes during the run rather than from a favourable starting point. Against the PyTorch defaults the median is 2.8×. Counting every seed separately, Paramorph matched or beat the baseline within the training budget in 95% of all evaluation runs. The Hugging Face Trainer defaults are tuned for fine-tuning rather than pre-training from scratch, which explains the much larger speed-ups against them; we include them because they are what many practitioners start from.

Baseline	Tuning compute per task	Tasks evaluated	Tasks with a speed- up	Median speed-up	Interquartile range	Tasks ≥ 2×
FIXED CONFIGURATIONS						
RecurviaDefaults	none	41	41	4.0×	2.2–7.4×	35
RecurviaDefaultsScaled	none	41	41	5.4×	3.8–16×	39
TorchDefaults	none	41	39	2.8×	1.9–4.2×	29
HuggingFaceDefaults	none	41	41	23×	16–46×	41
HuggingFaceDefaultsConstantLR	none	41	41	14×	9.5–26×	41
PER-TASK SEARCHES						
TunedLR	256 training runs	7	7	2.1×	1.3–9.7× (range)	4
TunedFull	256 training runs	7	3	1.0×	1.0–2.1× where recorded	1

Table 1. Speed-ups over the 41-task suite. Each task’s value is the mean over the seeds, of five, that reached the baseline’s best loss; a task with no speed-up is one where no seed did so within the budget, and a 1.0× tie is not counted as a speed-up; such tasks are counted at 1× when computing medians. Medians are used because a few tasks give very large ratios (above 100×) when the baseline makes little progress within its budget. The tuned baselines were run on seven tasks because each costs 256 training runs; they are compared in detail in Section 6. Per-task values are listed in Appendix C.

The tuned baselines are a much harder test, since each represents 256 training runs devoted to a single task, and the per-task search has the advantage of selecting the configuration with hindsight. Paramorph reaches the learning-rate-tuned baseline’s best loss sooner on all seven tasks on which it was run, by 1.3× to 9.7×, and reaches the fully tuned baseline’s best loss sooner on three of seven, by up to 2.1×. Where it does not, it is often close, and the relevant comparison is how much search it takes to beat a single Paramorph run, which Section 6 measures directly.

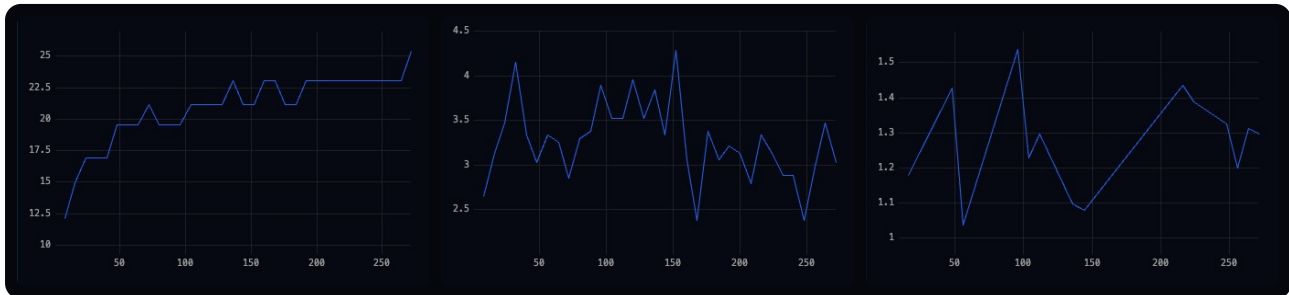
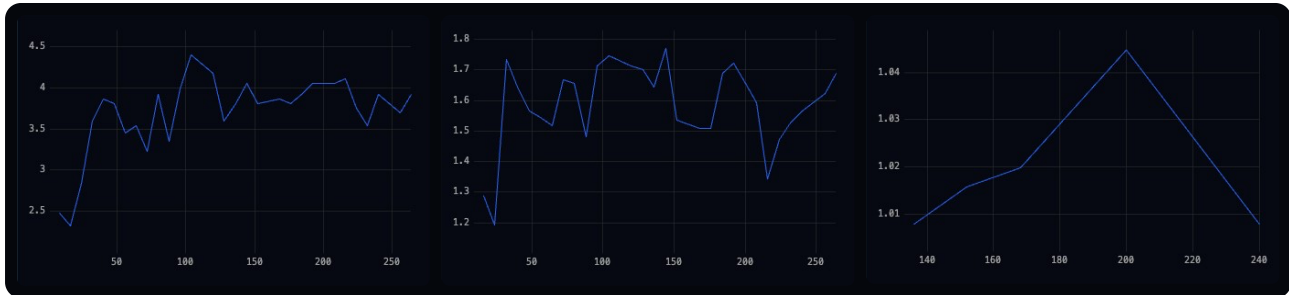
5.2 Learning progress during training

Figure 5 tracks the speed-up on two tasks at successive Paramorph checkpoints as the policy trains (all tasks: [training-eval runs in Mettrana](#)). On `gpt2-nano` the speed-up against the Hugging Face defaults roughly doubles over the course of training, from about 12× to about 25×, consistent with the policy learning transferable behaviour rather than fitting its training tasks. Against `RecurviaDefaults` the speed-up rises over the early checkpoints and then fluctuates, between about 2.4× and 4.3× on `gpt2-nano` and 1.2× and 1.8× on `gpt2-off_c4`. Against the fully tuned baseline, individual single-seed evaluations already beat the 256-trial search at a number of checkpoints, by up to 1.5× on `gpt2-nano` and by 1.01–1.05× on the larger `gpt2-off_c4`. With five seeds at the late checkpoint used in Section 6, the full search overtakes Paramorph on `gpt2-nano` after 67 to 164 trials, so against this baseline the picture is one of close competition rather than a consistent win.

vs HuggingFaceDefaults

vs RecurviaDefaults

vs TunedFull (256 trials)

Top row: `gpt2-nano-global_wikipedia`

— Speed-up of a single-seed evaluation run at each Paramorph checkpoint

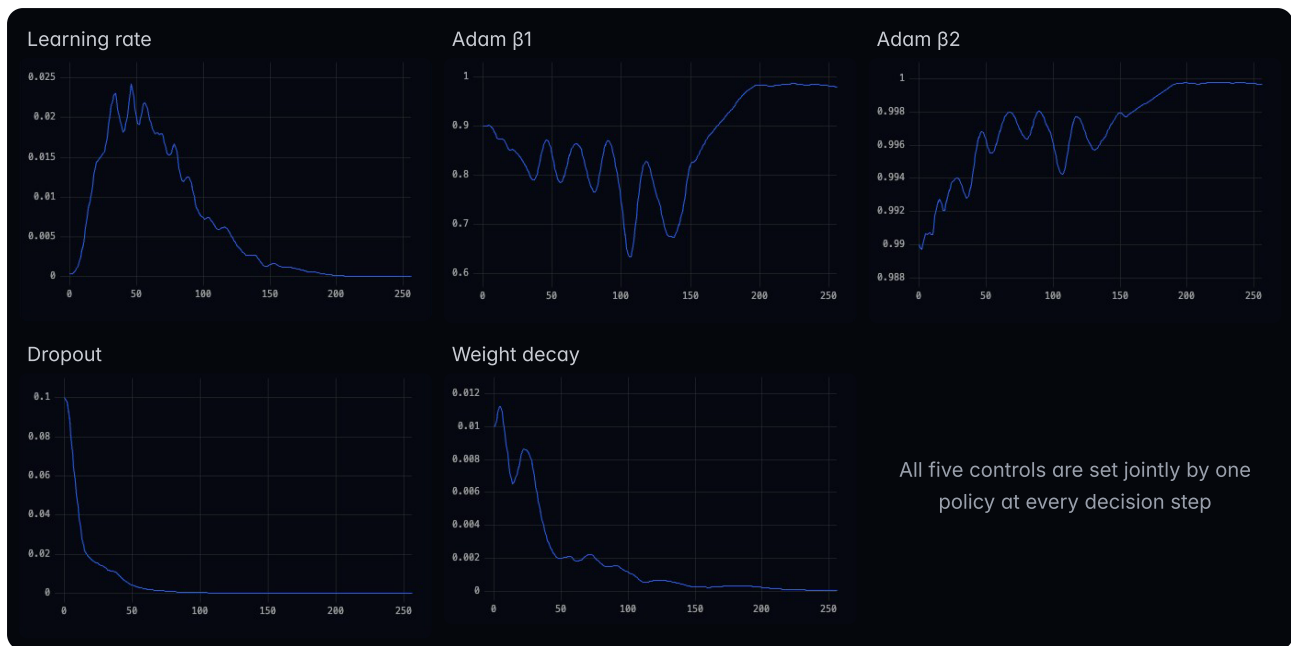
Figure 5. Speed-up as Paramorph trains. Bottom row: `gpt2-off_c4`, the official 124M-parameter GPT-2. Horizontal axes: Paramorph training progress. Vertical axes: speed-up. Gaps in the right-hand panels are checkpoints at which the tuned baseline was not beaten; the left and centre panels show a speed-up at every checkpoint.

5.3 What the policy learned

The schedules Paramorph produces are informative in their own right, because each one is the policy's solution to a specific run rather than a fixed recipe. Figures 6 and 7 show the full set of five controls on the two detailed tasks. Several features recur, and each corresponds to practice that experienced practitioners arrive at by hand:

- **Warm-up then decay.** The learning rate rises, peaks and decays on both tasks, a structure that was not specified anywhere and emerged from the reward.
- **Scale-aware learning rates.** The peak learning rate on the official 124M GPT-2 (about 0.0034) is roughly seven times lower than on the much smaller `gpt2-nano` (about 0.024), in line with the standard practice of lowering the learning rate as models grow.
- **No dropout for single-pass pre-training.** Dropout falls from 0.1 to below 0.01 within about 50 decision steps on both tasks and to essentially zero thereafter, matching modern language-model pre-training, where each token is seen only once and dropout is usually disabled.
- **Annealed weight decay.** Weight decay is reduced towards zero over the run on both tasks.
- **Task-specific momentum.** Adam β_1 moves in opposite directions on the two tasks: it rises to about 0.98 late in training on `gpt2-nano` and falls to about 0.1 on the 124M GPT-2. The coordinated changes in learning rate and momentum are consistent with the multi-agent design, in which every agent optimises the same shared objective.

Combined with the seed-to-seed variation in Figure 4, this shows a controller that responds to the run in front of it, which a pre-computed schedule cannot do.



— Value chosen by Paramorph

Figure 6. Learned schedules on `gpt2-nano-global_wikipedia`. Horizontal axes: decision steps of 100 batches over the 256-step budget.



— Value chosen by Paramorph

Figure 7. Learned schedules on `gpt2-off_c4` (official 124M-parameter GPT-2, trained on C4). Horizontal axes: decision steps of 100 batches.

6 Paramorph against tuned baselines

Comparisons with fixed configurations are like-for-like, since neither side spends compute on tuning. A tuned baseline is different: it is the outcome of a search in which every trial is a training run. The question a practitioner faces is therefore not only whether a tuned configuration eventually beats Paramorph, but **how many training runs it takes to get there**. A single Paramorph run that matches a well-tuned baseline replaces the search entirely.

To answer this, we compare the best validation loss of each Paramorph seed with the running best of the Optuna search as trials accumulate. The point where the search curve crosses Paramorph's results is the amount of tuning compute that one Paramorph run is worth.

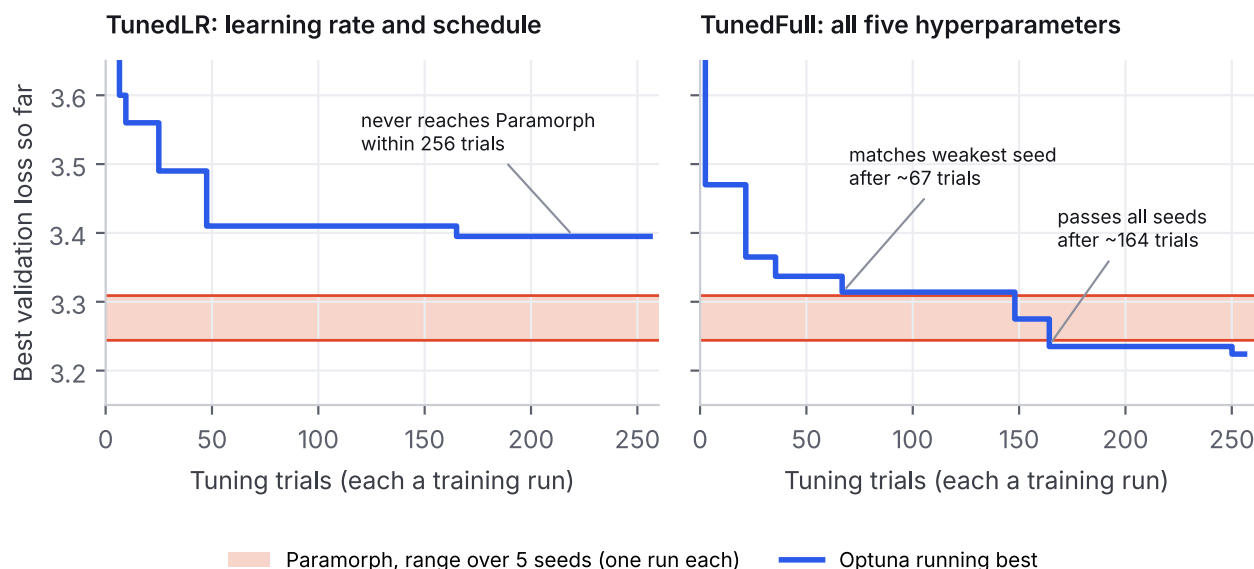


Figure 8. Tuning compute needed to match one Paramorph run on `gpt2-nano-global_wikipedia`. The shaded band spans the best validation loss of five Paramorph seeds, each a single zero-shot run. The vertical axis is cropped to the region of interest; the first trials of each search start above it.

On `gpt2-nano` (Figure 8), the learning-rate search never reaches Paramorph within its 256 trials: every one of Paramorph's five seeds finishes below the best configuration it found. The joint search over all five hyperparameters, a much larger search space, needs about 67 trials to match Paramorph's weakest seed, and about 164 trials to pass all five. In other words, a single Paramorph run delivered what would otherwise take several dozen to well over a hundred training runs of a modern hyperparameter optimiser, with the variation across seeds setting the exact figure.

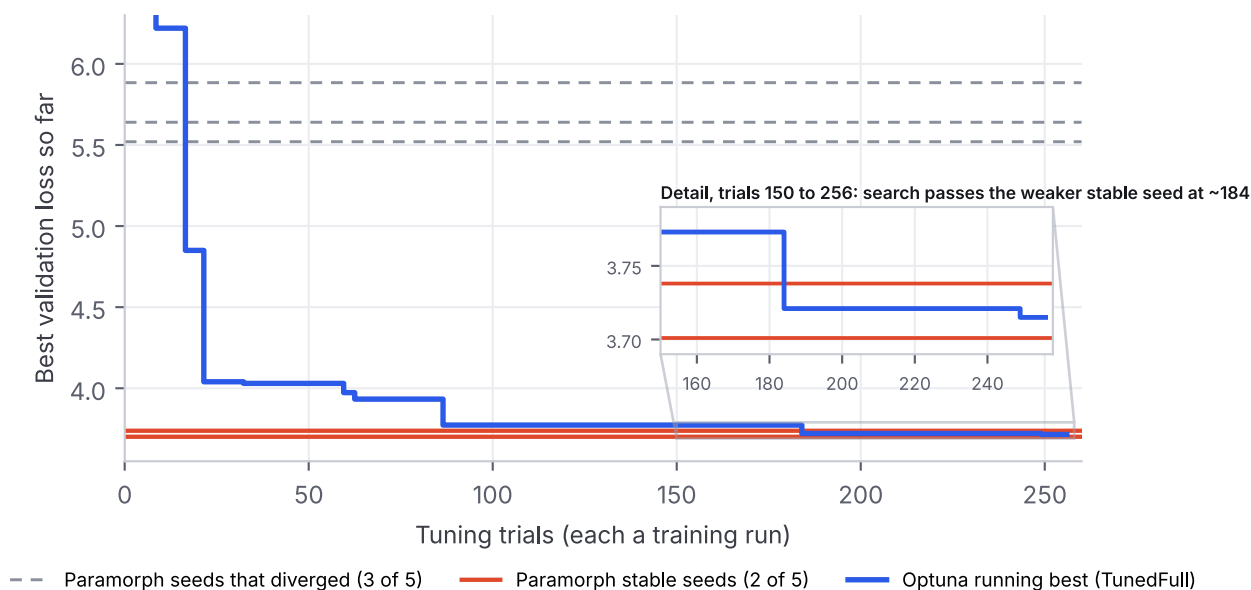


Figure 9. Tuning compute needed to match Paramorph on `gpt2-off_c4` (official 124M-parameter GPT-2). Lines show the best validation loss of each of five Paramorph seeds; three seeds diverged (Section 7.1). The inset magnifies the final stage of the search.

The official 124M-parameter GPT-2 (Figure 9) is a stringent test: it lies well outside Paramorph’s training distribution, and the fully tuned baseline is strong. Three of the five seeds diverged, a limitation we discuss in Section 7.1. The two stable seeds, however, reached validation losses of about 3.74 and 3.70. The 256-trial search only passed the first after about 184 trials, and its final best of 3.715 is essentially level with the second. This is consistent with the 1.01–1.05× speed-ups over this baseline seen during training (Figure 5).

THE COMPUTE ARITHMETIC

A tuned baseline costs up to 256 training runs per task, less the saving from early pruning, and must be repeated whenever the model, data or budget changes. Paramorph costs one run per task, and its own training cost is incurred once and amortised over every run it subsequently controls, and the policy itself is small and is queried only once every 100 batches. Each crossover point above comes from a single search per task, and Optuna runs trials in parallel with pruning, so the exact trial counts carry some noise; the scale of the saving does not depend on them. Matching a tuned result is therefore already a large saving; the zero-shot gains over untuned configurations in Section 5 come on top of avoiding the search.

For the largest models the comparison changes character, because a multi-run search is no longer possible at all: no organisation can afford hundreds of full training runs of a frontier-scale language model. Practitioners instead tune on a series of smaller models and extrapolate the best settings to the target scale using fitted scaling laws or a parametrisation such as μP . Each step of that process adds uncertainty. The small-scale optima are themselves noisy estimates, the fitted trends are extrapolated well beyond the range they were measured on, and the final run at full scale cannot be repeated if the extrapolated settings prove suboptimal. A controller that adapts the hyperparameters during the one run that matters addresses exactly this gap.

7 Generalisation and reliability

A training controller is only useful if it works on workloads it was not trained on. We examine five axes along which a deployed workload can differ from the training distribution: model size, model architecture, training length, batch size and context length. Changes of dataset accompany the model-size comparison but are not isolated here. We also examine reliability, meaning how consistent the speed-up is across seeds. Each task is evaluated with five seeds and, where possible, tasks in a comparison differ only along the axis being studied. Unless stated otherwise, speed-ups are measured against

`RecurviaDefaults`, which isolates Paramorph’s contribution most directly. The five-seed runs for every task, including those not shown here, can be inspected in Metrana: [multiseed-eval runs](#).

7.1 Model size

Larger models are under-represented in Paramorph’s training distribution because they are expensive to train at the scale reinforcement learning requires: almost all training tasks have fewer than 10M parameters, and the largest has 65M (dashed lines in Figure 10). Figure 10 shows the GPT-2 family from about 5M parameters to the official 124M model. Speed-ups are 3.6–3.9× at the smallest sizes, 2.2× at 26M and 1.7× at 124M, and against the 256-trial learning-rate-tuned baseline they remain between 1.3× and 2.2× at every size. All but the smallest models in this comparison are also trained on a different dataset (C4 rather than Wikipedia). Earlier evaluations have shown strong generalisation across datasets, so we attribute the decline mainly to model size, which is the axis we are now prioritising in training.

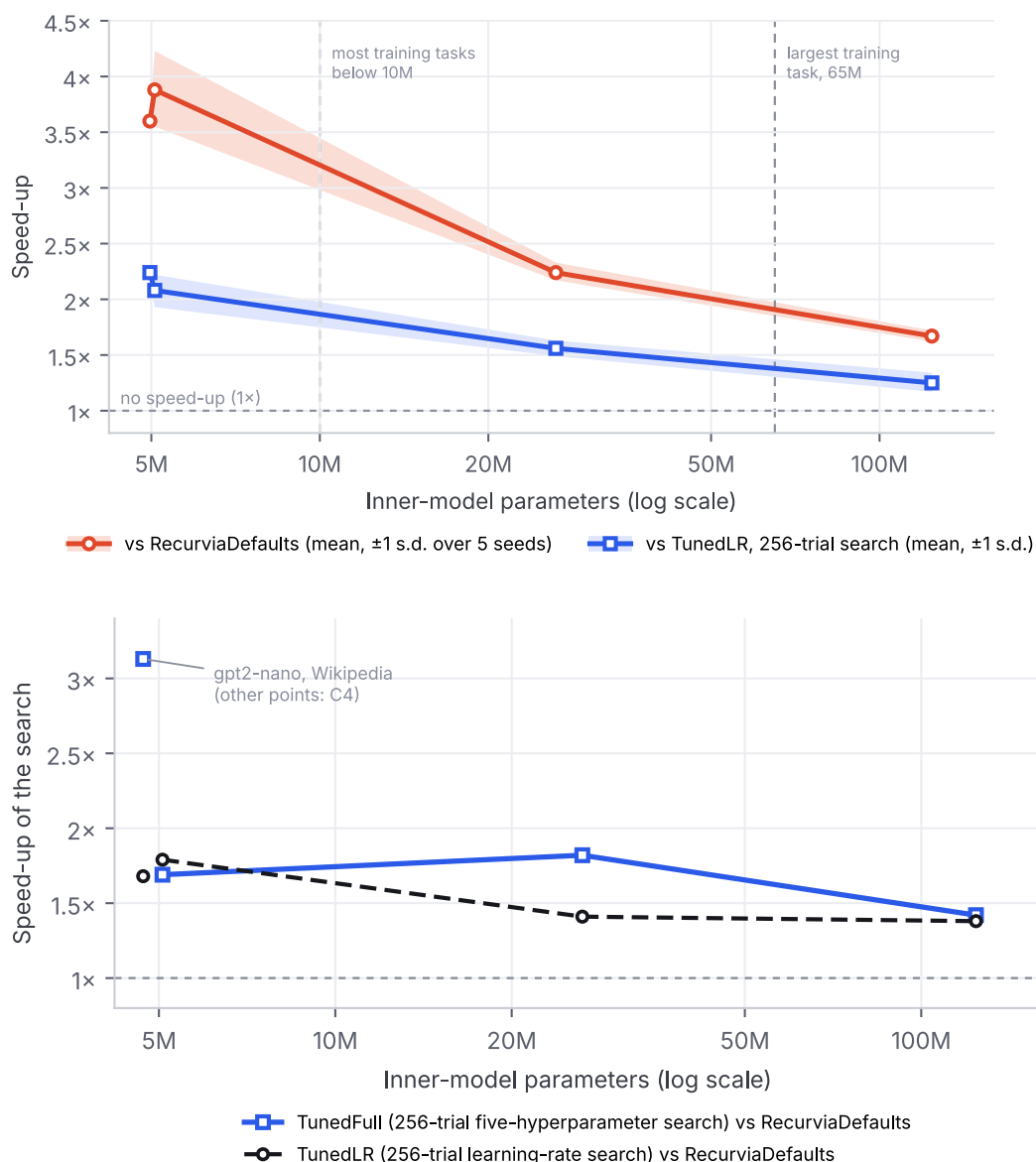


Figure 10. Speed-up across GPT-2 model sizes. Top: Paramorph’s speed-up. Each point is a task; the band is ± 1 standard deviation over five seeds. Against both the untuned and the 256-trial learning-rate-tuned baseline, every size shown has a speed-up; at 124M parameters the speed-up comes from the two seeds that remained stable. Bottom: the speed-up that the 256-trial searches themselves achieve over `RecurviaDefaults` on the same tasks, a measure of how much there is to gain over the constant configuration.

How much speed-up is available on a task is not known in advance, which makes trends like this hard to interpret. One reference point is how much the tuned searches themselves gain over the same constant configuration (Figure 10, bottom). That gain also shrinks with model size, from 1.7–3.1 $\times$ on the smallest models to about 1.4 $\times$ at 124M, so part of Paramorph’s decline reflects less headroom over the constant configuration on larger models. This does not remove the need to address the instability described below. A modest speed-up on a large model also saves more compute in absolute terms than a large speed-up on a small one: Section 7.7 shows that the training time saved per run is in fact largest at 124M.

At 26M parameters (`gpt2-small-global_c4` , Figure 11) all five seeds are stable and closely grouped. They reach the baseline’s best loss after about 110 to 120 decision steps and finish around 6% lower in validation loss than the baseline.

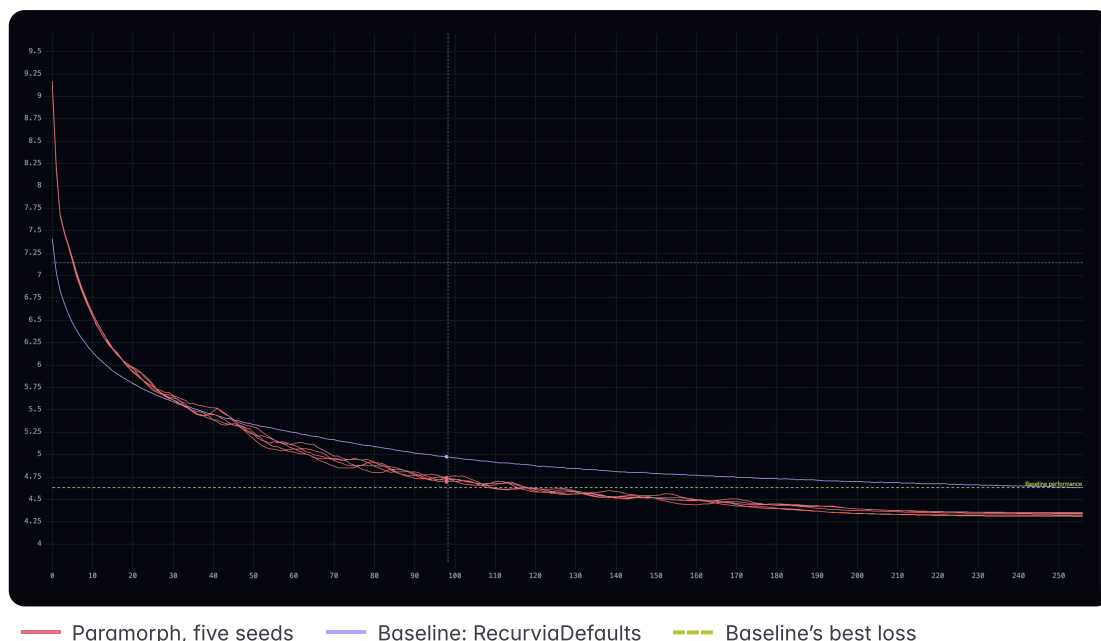


Figure 11. `gpt2-small-global_c4` (26M parameters). Validation loss against decision steps of 100 batches. The thin dotted cross-hair is a cursor marker from the dashboard.

The official 124M-parameter GPT-2 (`gpt2-off_c4`, 12 transformer layers, using the [published Hugging Face configuration](#)) is discussed separately here and shown in Figure 12. It lies well outside the training distribution, and here the policy is at the edge of its competence: three of the five seeds become unstable after about 25 decision steps and do not reach the baseline's best loss. The two stable seeds pass it at around step 150, a speed-up of about 1.7 $\times$, are 1.3 $\times$ faster than the 256-trial learning-rate search and go on to match the 256-trial fully tuned baseline (Section 6). The instability was not observed at 26M and, as Section 7.3 shows, is absent on OLMo 3 models up to 1B parameters, which suggests it is specific to this configuration rather than to model size alone. The team has several hypotheses for its cause and is running targeted experiments to test them (Section 8).

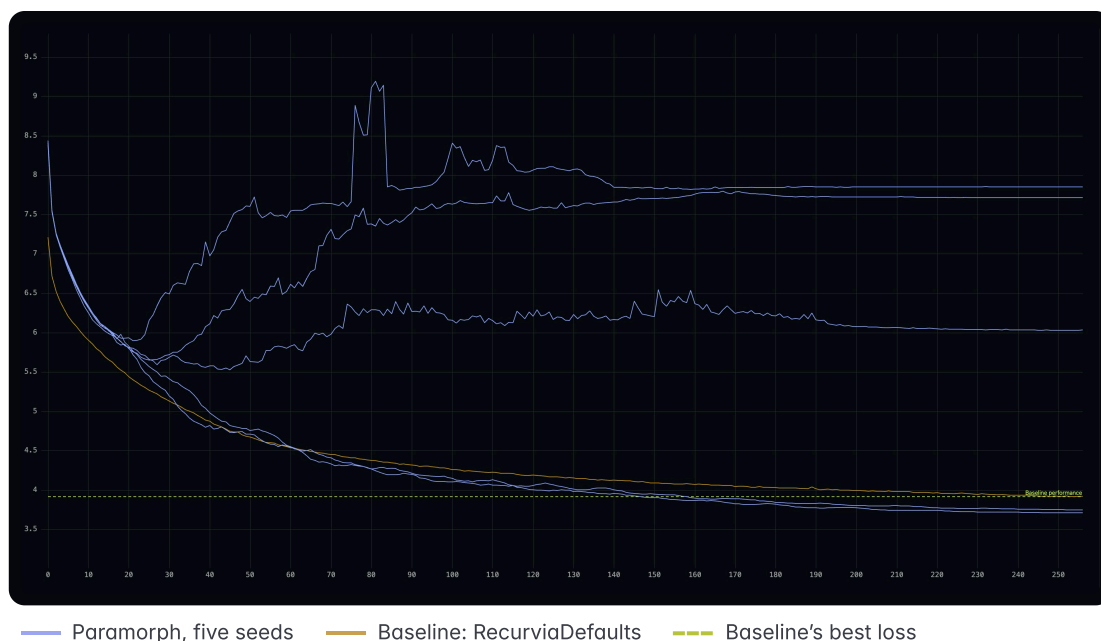


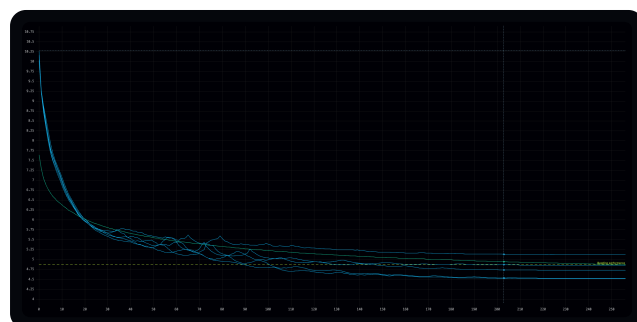
Figure 12. `gpt2-off_c4` (official 124M-parameter GPT-2). Two seeds train stably and finish clearly below the baseline; three diverge. Validation loss against decision steps of 100 batches.

7.2 Model architecture

Paramorph is trained on a distribution of architecture families, including GPT-2, SmoLM2 and Qwen3. Within each family, the training tasks define the model configuration explicitly, in terms of the number of layers, the number of attention heads and the hidden dimension, and these are varied considerably across the training distribution, so the policy sees many different shapes of model rather than a handful of published checkpoints. It is evaluated on held-out tasks from those families as well as on families it has never seen. Figure 13 shows four cases. On the held-out `qwen3-small` task, all five seeds reach the baseline's best loss by about step 50, a speed-up of roughly $5\times$, and finish about 15% lower in loss. SmoLM2 shows the widest spread of the four, with three seeds finishing clearly below the baseline, one level with it and one above. On `qwen3-base`, which is larger than any Qwen3 model in training, all five seeds still pass the baseline, at a speed-up of about $1.9\times$.

The OLMo family is entirely absent from training, and every OLMo task is out of distribution on three separate counts: the model architecture, the tokenizer used to preprocess its data, and the training framework, since all of Paramorph's training tasks use the Hugging Face Trainer while OLMo models are trained with their own. On an OLMo 2 model whose size is within the training range, all five seeds pass the baseline's best loss by about step 60, a speed-up of roughly $4\times$, and finish about 7% lower, with very little variation between seeds.

SmoLM2 (held out, in distribution)



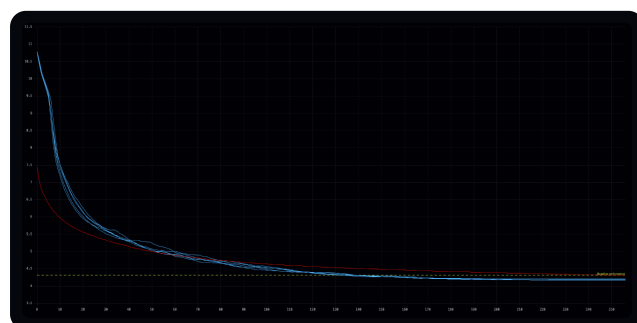
— Paramorph, 5 seeds — Baseline

Qwen3 small (held out, in distribution)



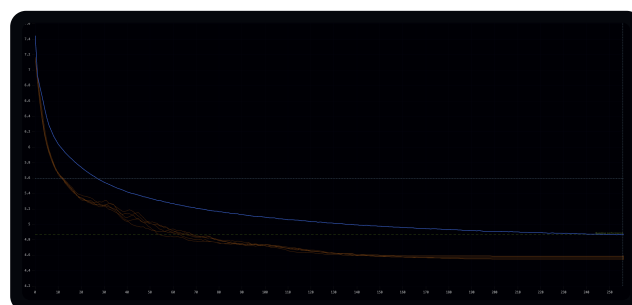
— Paramorph, 5 seeds — Baseline

Qwen3 base (out of distribution: model size)



— Paramorph, 5 seeds — Baseline

OLMo 2 tiny (out of distribution: model family)



— Paramorph, 5 seeds — Baseline

Figure 13. Generalisation across architecture families. Validation loss against decision steps of 100 batches, all trained on C4. Baseline: `RecurviaDefaults`; the dashed horizontal line in each panel marks the baseline's best loss.

7.3 Scaling to 1B parameters on an unseen family

The OLMo 3 evaluation is a demanding test of generalisation. Every OLMo 3 task is out of distribution by architecture, tokenizer and training framework. Even the smallest is near the upper edge of the sizes Paramorph was trained on, and from 100M parameters upwards every model is larger than any training task; the largest has around 1B parameters. Figure 14 shows two views of the same runs.

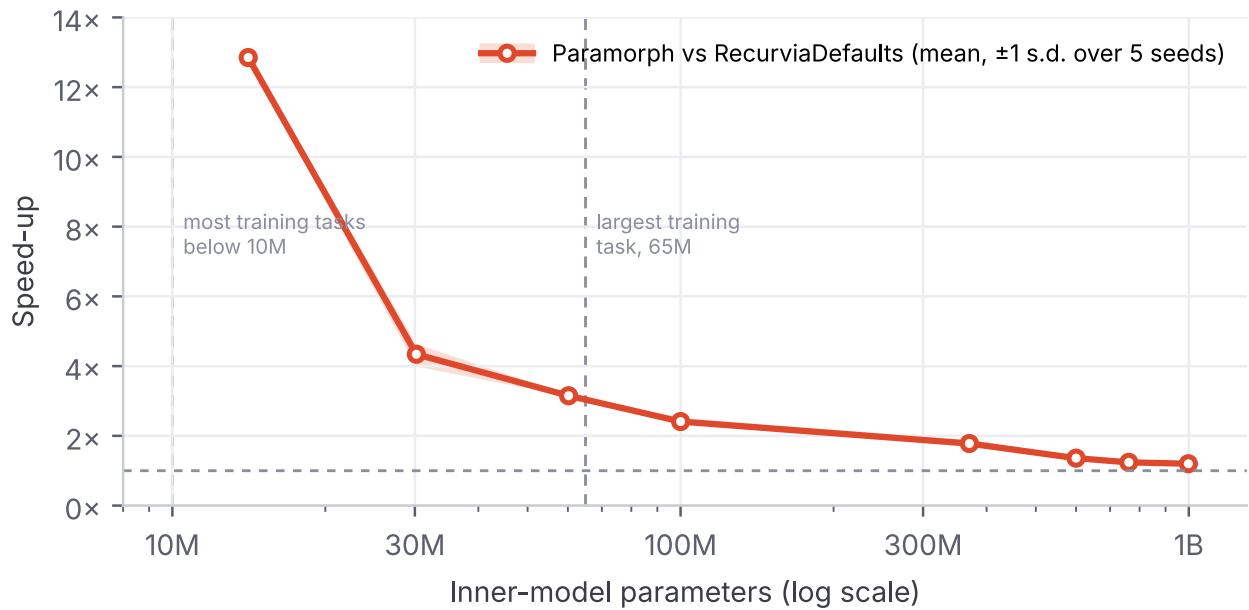


Figure 14. OLMo 3 from about 14M to 1B parameters, a model family absent from training. Speed-up against RecurviaDefaults. Each point is a task; bands show ± 1 standard deviation over five seeds. Dashed lines mark the practical limits of the training distribution.

Against Paramorph's own constant starting values, the speed-up is large for small models, 12.9 $\times$ at 14M parameters, and declines with size to 1.2 $\times$ at 1B, a pattern similar to the GPT-2 results; the time saved per run, however, remains substantial at every size (Section 7.7). Paramorph improves on its starting point at every size, on a family it has never seen.

Unlike the official 124M GPT-2, the OLMo 3 runs show no instability, even at 1B parameters (Figure 15). Paramorph's loss starts higher than the baseline's early in training and then converges to the baseline's best loss ahead of the end of the budget.

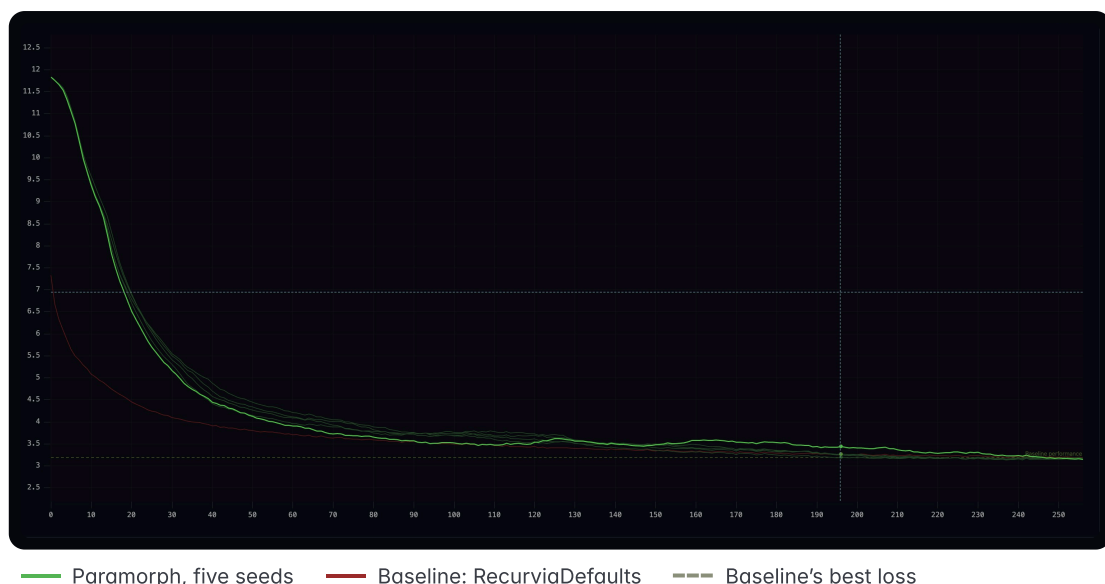


Figure 15. OLMo 3 at about 1B parameters. Validation loss against decision steps of 100 batches. All five seeds train stably. The cross-hair is a dashboard cursor marker.

7.4 Training length

We next vary the training budget on `gpt2-nano-global_wikipedia`, from 16 to 4,096 decision steps. The training budget is one of Paramorph’s inputs, so the policy can plan its schedule around it. Almost every training run of Paramorph was shorter than 1,000 decision steps, so the three longest budgets (1,024, 2,048 and 4,096 steps) lie beyond its training experience in practice. There is a speed-up at every budget: about 2× for short budgets, rising steeply for longer ones to 7.4× at 1,024 steps, 13.2× at 2,048 and 24.8× at 4,096 (Figure 16). Part of this increase reflects the fixed baseline converging well before the end of long budgets (Figure 17): with a constant configuration, extra steps buy the baseline very little, while Paramorph continues to use them. A constant configuration cannot adapt to a longer budget at all; schedule-based recipes can, but must be re-tuned for each horizon, whereas Paramorph adapts from the budget input alone.

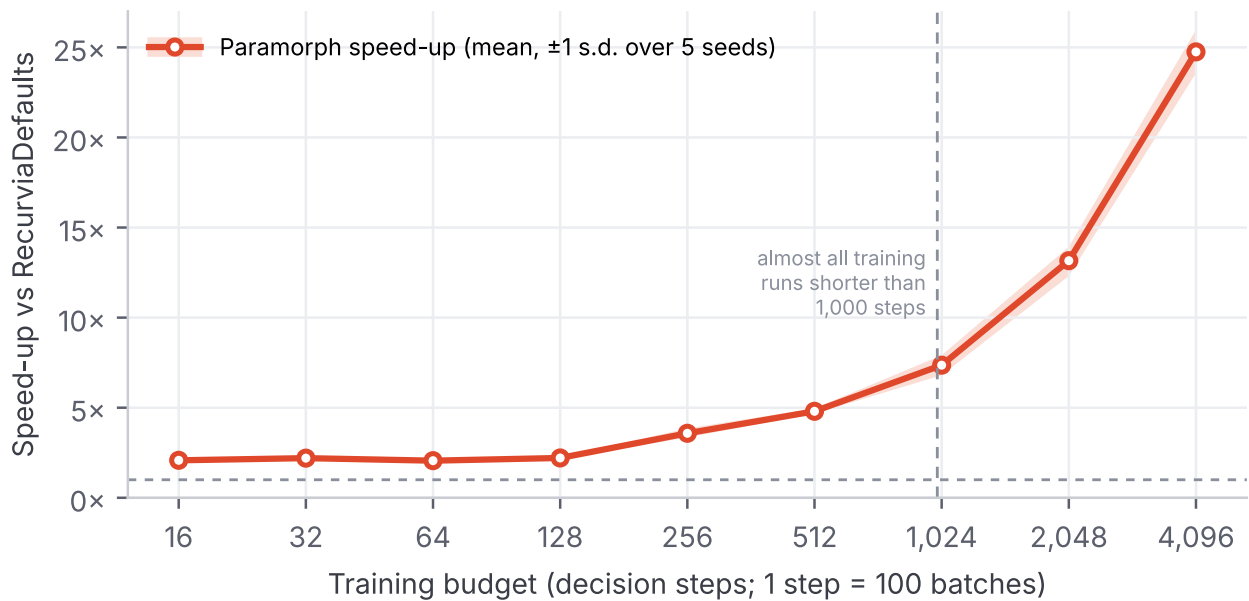


Figure 16. Speed-up as a function of training budget on `gpt2-nano-global_wikipedia`. The standard evaluation budget elsewhere in this report is 256 steps. The dashed line marks the practical limit of the training distribution.



Figure 17. The 4,096-step budget. Paramorph reaches the baseline's best loss in under 200 steps and finishes about 3% lower in loss; the fixed baseline improves only slowly over the remaining budget. Horizontal axis: decision steps of 100 batches.

The budget input also visibly shapes the policy's behaviour. Figure 18 shows one seed at each of four budgets. Shorter budgets anneal the learning rate earlier, and the runs given longer budgets go on to finish lower: the 512- and 1,024-step runs both end below the 256-step run. A stricter property would be for each run to hold the lowest loss of all runs still in progress at the moment its own budget ends, so that a shorter run trades long-term progress for the best possible result at its deadline. The current policy does not yet do this consistently: at step 256, for example, the 256-step run sits slightly above the runs with longer budgets. Table 2 quantifies the property across all budgets. For each budget, it gives the share of comparisons with other budgets that are correctly ordered: a run should be at or below the loss of every longer-budget run at the moment its own budget ends. A random ordering scores 0.5 and perfect budget-awareness scores 1.0. The extrapolated score also includes shorter runs, by holding their loss constant after they finish. Ordering is close to perfect for budgets of 64 and 128 steps, but for budgets of 256, 512 and 2,048 steps the unextrapolated score falls below the 0.5 expected from random ordering (0.45, 0.21 and 0.24). Sharpening this budget-awareness is a defined objective for the next training runs.

Training budget (decision steps)	Ordering score	Extrapolated score
16	0.835	0.835
32	0.880	0.895
64	0.993	0.995
128	0.992	0.995
256	0.450	0.725
512	0.213	0.650
1,024	0.620	0.855
2,048	0.240	0.750
4,096	—	0.870
Mean	0.653	0.841

Table 2. Budget-awareness on `gpt2-nano-global_wikipedia`. The 4,096-step run has no longer run to compare with, so only the extrapolated score is defined.

Validation loss



Learning rate (log scale)



One seed per training budget of 256, 512, 1,024 and 2,048 decision steps; each line ends where its budget ends.

Figure 18. Budget-aware schedules. Left: validation loss; the 2,048-step run is shown up to step 1,024. Right: the learning rate chosen for each budget. Shorter budgets begin their decay earlier.

7.5 Batch size

Batch sizes above 512 are fully outside Paramorph’s training distribution, and even within it the larger sizes are under-represented. Varying the batch size of `gpt2-nano-global_wikipedia` from 8 to 2,048 (Figure 19), there is a speed-up at every batch size, rising steadily from 2.2× to 8.4× at 256 and continuing to rise outside the training distribution, to 11.4× at 1,024 and 15.7× at 2,048. Because the reference configuration is constant and its learning rate is not rescaled with batch size, part of this growth reflects the baseline becoming less well matched as batches grow; the policy, by contrast, adapts its schedule to the batch size it observes. Larger batches are the regime in which large-scale training operates, so generalisation along this axis is of real practical importance.

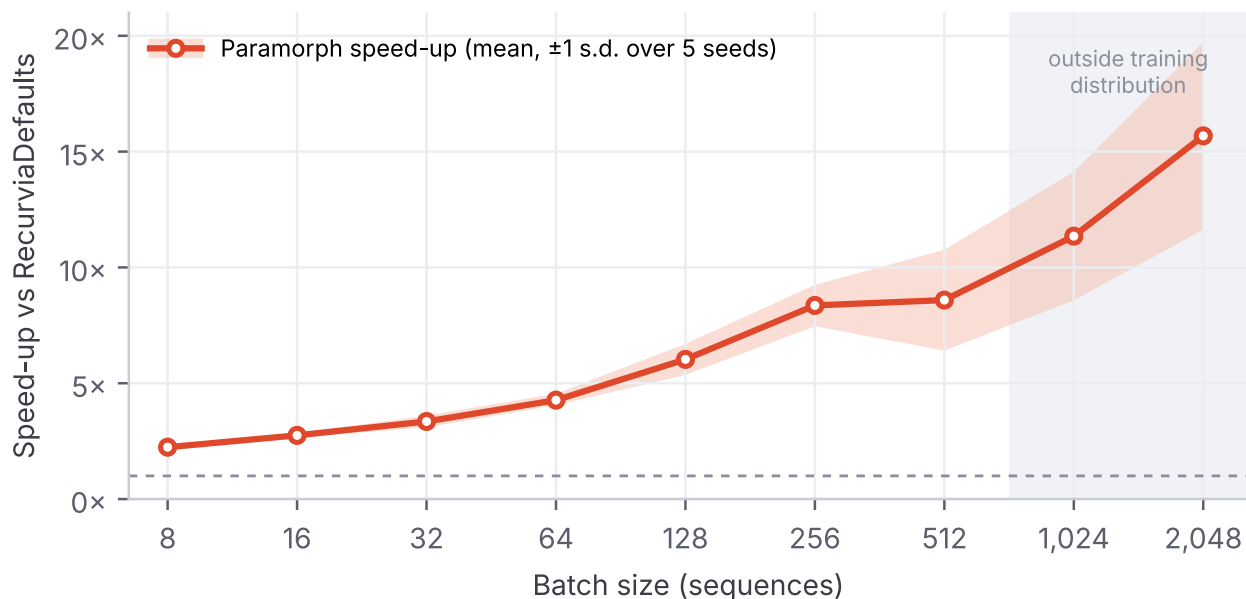
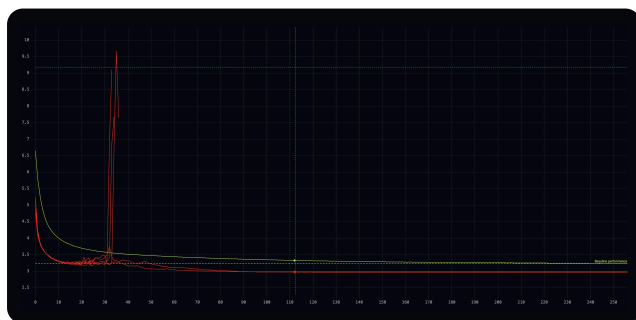


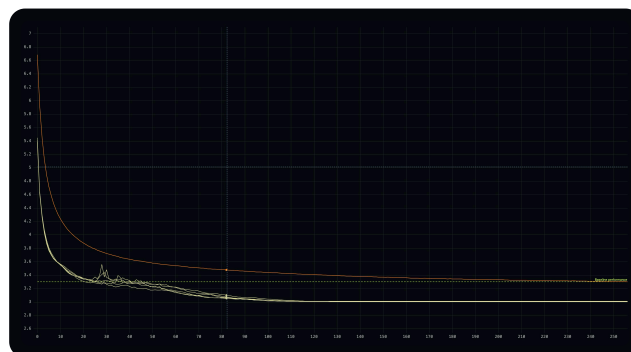
Figure 19. Speed-up as a function of batch size on `gpt2-nano-global_wikipedia`. The shaded region lies outside the training distribution.

At the two largest batch sizes some seeds become unstable, as shown for a batch size of 2,048 in Figure 20. Even these seeds pass the baseline's best loss before they diverge, so the measured speed-ups are real, and the remaining seeds train stably and finish well below the baseline. At a batch size of 512 no seed diverges (Figure 21).



— Paramorph, 5 seeds — Baseline

Figure 20. Batch size 2,048, out of distribution. Some seeds diverge after step 30, having already passed the baseline's best loss.



— Paramorph, 5 seeds — Baseline

Figure 21. Batch size 512. No seed diverges; the baseline's best loss is reached around step 30, roughly 8× sooner.

7.6 Context length

All context lengths tested lie within the training distribution, although the longer ones are under-represented. The speed-up is 4.8× at 128 tokens and between 7.0× and 7.5× from 256 to 1,024 tokens (Figure 22), so performance holds as sequences lengthen. At a context length of 1,024 one seed shows a late loss increase, but it does so well after passing the baseline's best loss, so the speed-up is preserved, and the instability is milder than at large batch sizes (Figure 23).

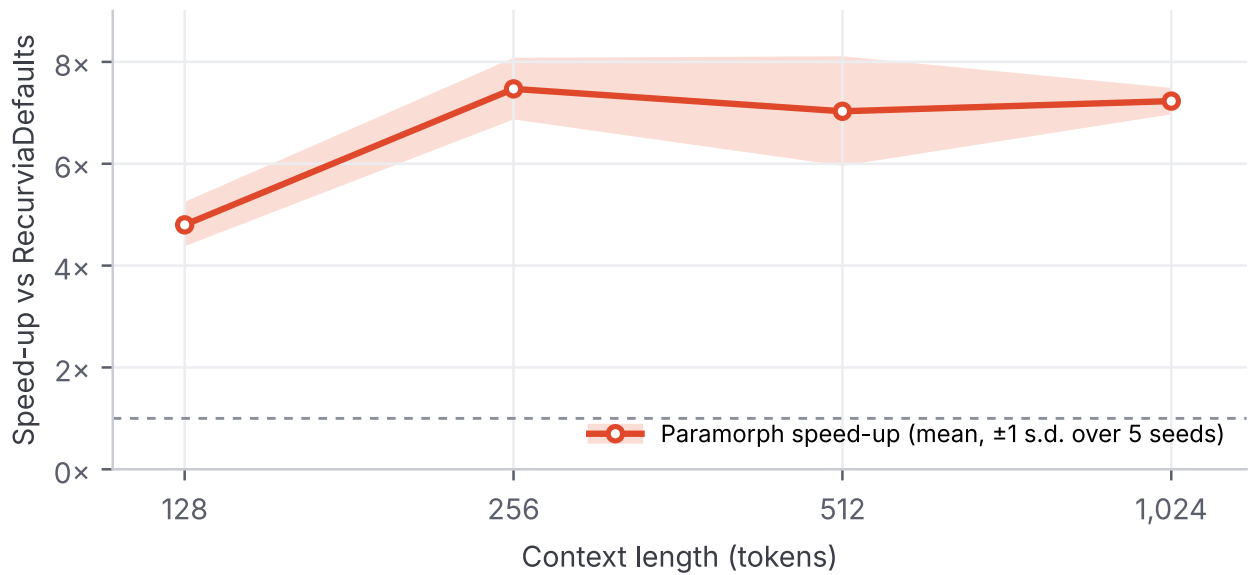


Figure 22. Speed-up as a function of context length on `gpt2-nano-global_wikipedia`.

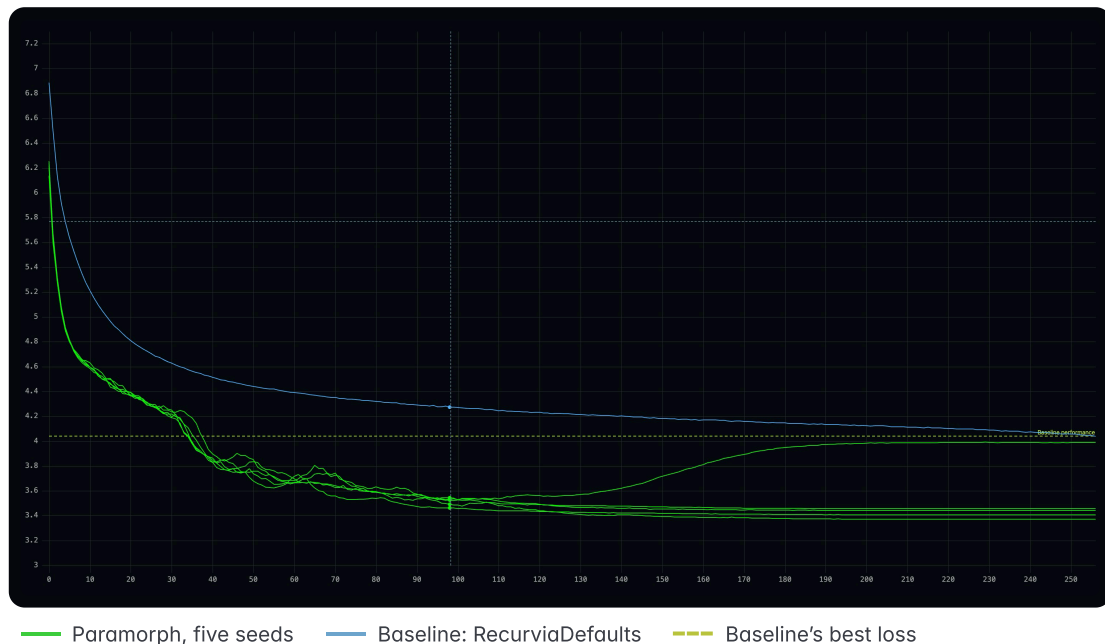


Figure 23. Context length 1,024. Paramorph passes the baseline's best loss by about step 35. One seed drifts upwards late in training but remains at or below the baseline's best loss. Horizontal axis: decision steps of 100 batches.

7.7 Absolute savings as models grow

The speed-up ratios in Sections 7.1 and 7.3 decline as models get larger. That does not mean the benefit shrinks. Each training step of a larger model costs more compute and time, so a smaller ratio can still save more in absolute terms. Figure 24 converts the measured speed-ups into training time saved per run, combining each task's step savings with its measured cost per step.

On the GPT-2 family the saving against `RecurviaDefaults` is largest at the largest model: about 99 minutes per run on the official 124M GPT-2, where the speed-up ratio is 1.7 $\times$, compared with 65–76 minutes at about 5M parameters, where the ratio is 3.6–3.9 $\times$. Against the 256-trial learning-rate search the pattern is the same: the largest saving, about 38 minutes per run, is again at 124M, where the ratio is

1.3×. That comparison also leaves out the cost of the search itself, which would add hundreds of training runs to the baseline's side. Across the GPT-2 tasks with tuned baselines, the learning-rate search did not catch up with any stable Paramorph seed.

On OLMo 3, a family Paramorph never saw in training, the saving per run stays between about 65 and 235 minutes from 30M to 1B parameters, even as the ratio falls from 4.3× to 1.2×. These figures understate the saving at the larger sizes: to fit the available compute, the larger OLMo 3 tasks used smaller batches than are recommended in practice, which lowers their cost per step. At recommended batch sizes, each step would cost more, and if the relative results held, the absolute saving would be considerably larger.

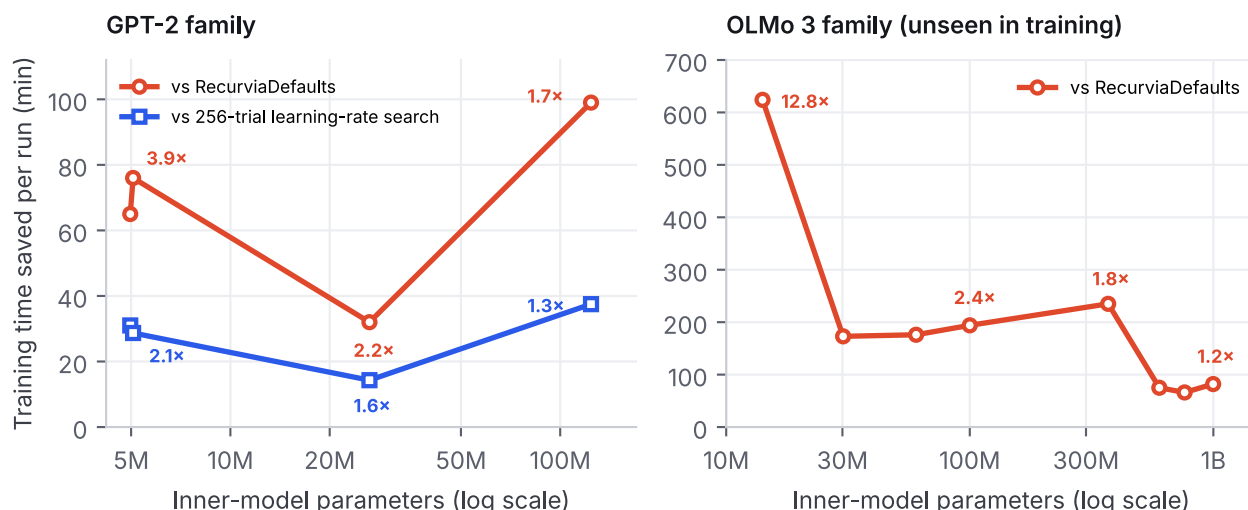


Figure 24. Training time saved per run as model size grows. Averaged over seeds; the labels give the corresponding speed-up ratio. Left: GPT-2 family against `RecurviaDefaults` and against the 256-trial learning-rate search (excluding the cost of the search itself). Right: OLMo 3 against `RecurviaDefaults`; larger OLMo 3 tasks used smaller-than-recommended batches, which understates their absolute saving. Times are measured on Recurvia's evaluation hardware and read from Recurvia's analysis charts, so are approximate.

Why the ratio declines. In these evaluations every inner task is trained on the same amount of data, whatever its size. Larger models need proportionally more data to train well (compute-optimal training scales the number of tokens roughly in proportion to parameters, as in the Chinchilla scaling results), so at a fixed data budget a larger model is effectively on a shorter run relative to what it needs. Section 7.4 shows that shorter runs leave less room for an adaptive schedule: on a fixed model the speed-up rises from about 2× on short budgets to 24.8× on long ones. We therefore expect the declining ratio to reflect the fixed data budget at least as much as model size itself. We are testing this directly with additional experiments, and future evaluations will scale each task's training data with model size towards the compute-optimal amount, so that larger models are compared on a like-for-like basis.

8 Current limits and next steps

The evaluation points clearly to where the current policy is weakest. For each, we have identified concrete next steps, several of which are improvements to the training method itself rather than simply more training.

Observation	Likely cause	Next step
Instability on some seeds on the official 124M GPT-2 and at batch sizes of 1,024 and above	Partly coverage: these regimes are absent from or under-represented in training, because large inner runs are expensive at reinforcement-learning scale. Partly design: we have identified specific improvements to the training signal and observations that target late instability.	Targeted experiments on reward, observation and optimisation design, alongside wider coverage of larger models and batches.
Speed-up ratio against fixed defaults declines with model size, although the absolute time saved per run does not (Section 7.7)	Most likely the fixed amount of inner training data, which leaves larger models on relatively short runs; also coverage of larger models in training.	Scale each task's training data with model size towards the compute-optimal amount, already under test; widen coverage of larger models.
Budget-awareness not yet fully exploited	The policy already responds to its budget input. We are planning further enhancements to its reward function, which we expect to address this.	Target this property explicitly in training and evaluation.
Not yet consistently ahead of a full 256-trial search	A per-task search is specialised to one task and selects with hindsight on it, whereas the foundation policy is a single general policy across all tasks.	Improve the general policy through the identified training changes and wider coverage. Where a partner needs to beat a tuned baseline on a specific, recurring workload, train a bespoke policy specialised to it (below). Single-task policies have already consistently beaten a fully tuned baseline (Figure B8).

Closing these gaps is not simply a matter of training on more and larger tasks. Wider coverage matters, and our internal review of the training stack has also identified a set of specific improvements to the policy's reward, observations and optimisation that target stability and budget use. None was applied in the run reported here, which gives the next training runs clear room to improve on these results. Each improvement needs to be validated carefully, at scale and against strong baselines, which makes the next stage experiment-intensive: it needs many training runs and enough researchers to design, run and interpret them. Compute is the smaller constraint. Recurvia is part of the European AI Factories programme and has already received 200,000 GPU hours through it, with further allocations expected. The binding constraint is the number of well-designed experiments that can be run and interpreted, which is why we are also working to automate parts of the research loop (below). Appendix B shows how the same team has worked through comparable problems over the past year, including the plateau on the 124M GPT-2 that was diagnosed and fixed in July 2026. The immediate priorities are:

- **Stability at larger scale.** Testing changes to the reward, observations and optimisation of the policy, together with wider training coverage of models at 124M parameters and above and batch sizes above 512, with the explicit target of eliminating the divergences seen in Sections 7.1 and 7.5.
- **Stronger comparisons at scale.** Tuned and μ P baselines on OLMo 3 at 1B parameters, extending the analysis of Section 6 to 1B parameters.
- **Budget-awareness.** Planned enhancements to the reward function so that each run finishes at its best within its budget, with final loss at budget reported alongside speed-up.
- **Per-layer control.** Resolving the multi-agent training details that currently make per-layer control less stable than global control on larger models, so that Paramorph can set a separate schedule for each

layer at scale.

Automating the research loop. The next stage is experiment-intensive, so we are also working to raise the number of experiments each researcher can run. Recurvia intends to begin running autoresearch workflows built around the agent in Metrana, its experiment-tracking platform. The agent can search across all of the logs from Paramorph's training and evaluation runs, both for the policy itself and for the inner training runs it controls, and will be able to edit code and launch new experiments automatically, with researchers setting the direction and reviewing the results. Developing this agent is part of Recurvia's roadmap: first its harness, meaning the tools, permissions and workflows it operates within, and soon afterwards post-training the underlying model with reinforcement learning so that it improves at this kind of research.

Bespoke policies for specific workloads. Alongside the general foundation policy, Paramorph can be trained on a single model or a narrow family of related workloads. In an earlier evaluation, a policy trained on one inner model matched or slightly exceeded the foundation policy on every reported measure for that model, and both beat a 256-trial search over all five controlled hyperparameters. Training a bespoke policy takes many training runs of its own, so it does not offer the single-run saving of the foundation policy on a one-off task. Its cost is justified for workloads that are trained repeatedly, as production teams do when they retrain on new data, where it is incurred once and spread across every subsequent run. Results from bespoke policies also inform the general model, by showing which workloads and regimes it most needs to cover.

Broader coverage. The evaluations here concern transformer language models. Earlier Paramorph versions were trained successfully on convolutional networks and multi-layer perceptrons, and extending the current foundation policy to those and other families is on the near-term roadmap.

9 Conclusion

Paramorph shows that the control of neural-network training can be learned once and applied zero-shot. Its clearest result is against conventional tuning: a single Paramorph run is worth dozens to well over a hundred trials of a 256-trial joint search, and in some cases it matches the full search outright. Across the 41-task suite it improves on untuned defaults, with a median speed-up of 2.8× over PyTorch defaults and a result that matched or beat the baseline in 95% of all evaluation runs, and it generalises to model families, sizes, batch sizes and training budgets beyond its training distribution, training stably on an unseen family up to 1B parameters.

The remaining gaps are stability on the largest GPT-2 configuration and the largest batches, and full use of the training budget. We have a clear plan for closing them: wider training coverage together with a set of already-identified improvements to how the policy is trained, validated through a sustained programme of experiments supported in part by compute from the European AI Factories programme. In the meantime, bespoke policies trained on specific, recurring workloads offer a way to bring the benefits of learned training control to those workloads directly.

A Appendix: definitions and experimental protocol

Inner task	A model, dataset and training configuration whose training Paramorph controls. Task names identify the model family, its size and the training dataset; for example, <code>gpt2-nano-global_wikipedia</code> is a small GPT-2 configuration trained on Wikipedia. The suffix <code>-global</code> , used in some figures, denotes global (rather than per-layer) control; the same tasks appear without it in Appendix C.
Inner model	The neural network being trained within an inner task, for example a GPT-2 or OLMo 3 language model. Its training hyperparameters are what Paramorph controls; Paramorph never sees its weights or training data, only summary statistics of how its training is progressing.
Outer model	The Paramorph policy: one actor-critic agent per controlled hyperparameter, trained with multi-agent reinforcement learning.
Decision step	One Paramorph decision, made every 100 batches of inner training. The standard budget is 256 decision steps, about 25,600 batches.
Speed-up	Let b be the step at which the baseline reaches its best validation loss within the budget, and p the first step at which Paramorph matches or improves on that loss. Speed-up = b / p . If Paramorph does not reach the loss within the budget, no speed-up is recorded. Both steps are the earliest at which the target is reached, so if the baseline diverges after reaching its best loss early, b is small and the speed-up is reduced; values of $1\times$ or below are possible in principle.
No speed-up	In the per-task results, a task shows no speed-up when Paramorph did not reach the baseline's best loss within the budget on any of its five seeds. A blank (—) means that baseline was not run for that task.
Seeds and checkpoints	Detailed results use five independent seeds per task with one Paramorph checkpoint taken from late in training, reported as mean $\pm$ one standard deviation. Per-task speed-ups are averaged over the seeds that reached the baseline's best loss. Progress during training is tracked by evaluating successive checkpoints on a single seed; not every checkpoint is evaluated on every seed.
Tuned baselines	Optuna, 256 trials per task, minimising final validation loss, with early pruning. <code>TunedLR</code> searches the learning rate and its schedule; <code>TunedFull</code> searches all five controlled hyperparameters jointly.
In distribution	A held-out task whose configuration (family, size, batch size, context length) could have occurred in training.
Out of distribution	A task outside the training distribution: all OLMo tasks and batch sizes above 512 could not have occurred in training, and models larger than any trained on lie outside it in practice.
Datasets	C4 (a large cleaned web-text corpus) and Wikipedia.
Software	<code>torch=2.11.0</code> , <code>transformers=5.5.4</code> , <code>olmo-core=2.5.0</code> .
Charts	Dark-background charts are taken from Metrana, with dashboard legends replaced by the legends shown beneath each figure; Figure 3 carries added annotations. Light-background charts were redrawn from the evaluation results in a consistent style.

Raw evaluation runs are publicly viewable in Metrana (Section 4.5), and per-task results are listed in Appendix C.

B Appendix: research progress, September 2025 to September 2026

Since September 2025 the research team has held an internal progress review every four to eight weeks. This appendix draws on those reviews to give a sense of how the programme has developed, the problems the team encountered and how it resolved them. It describes what was solved rather than exactly how; proprietary details of the training method are omitted. Figures from earlier stages were measured under the setups in use at the time and are not directly comparable with the results in the main report.

B.1 Timeline

2024	Single-model prototypes. Learning-rate control for convolutional networks, then learning rate and weight decay together, then transformer language models, with sample-efficiency gains of up to 2.65×.
Sep 2025	Building the foundation-model setup. On a 5M-parameter OLMo 2 task, the first version of the multi-task training setup reached a validation loss of 5.59 against 4.88 for a tuned baseline. The team began a systematic diagnosis.
Nov 2025	First stable learning. After fixing a bug in the open-source RL library and the main structural cause, Paramorph reached the tuned baseline on the same task for the first time (Figure B2).
Jan 2026	Faster iteration. Training throughput roughly doubled, and an “experiment funnel” of cheap proxy tasks was built so that design decisions could be tested quickly before committing to large runs.
Feb 2026	Systematic ablations. Controlled comparisons of the RL algorithm’s settings, the policy network and the action design settled the core configuration. An OLMo 3 trainer was integrated.
Mar 2026	A training signal that generalises. A new formulation removed the need for hand-set, task-specific targets. Paramorph trained stably on two very different tasks at once, then on a family of GPT-2 models, and per-layer control produced interpretable layer-by-layer schedules (Figure B4).
Apr 2026	First GPT-2 foundation-model run. Speed-ups of roughly 2× to 10× over Hugging Face defaults, beating tuned baselines on some tasks, with generalisation across model family, dataset, batch size and context length. The official 124M GPT-2 remained the hardest case.
May–Jun 2026	Scale-out. Multi-node runs on up to five B200 nodes, tuning of the RL setup, and a first sweep of the OLMo 3 model ladder from 1M to 1B parameters. A preliminary single-model OLMo 3 run at 30M parameters, using AllenAI’s own settings, matched the OLMo recipe at its standard training length and was roughly 3× faster at six times that length.
Jul 2026	Short-sightedness diagnosed and fixed. The cause of the plateau on the 124M GPT-2 was identified and corrected. On <code>gpt2-nano</code> , controlling six hyperparameters (the September run used five), Paramorph then consistently reached losses below a fully tuned baseline (Figure B8).
Sep 2026	Foundation-model run reported here. Improved control of the Adam momentum terms, training-length awareness and a final configuration study preceded the large-scale run launched on 13 September, evaluated in Sections 4 to 7.

B.2 Review by review

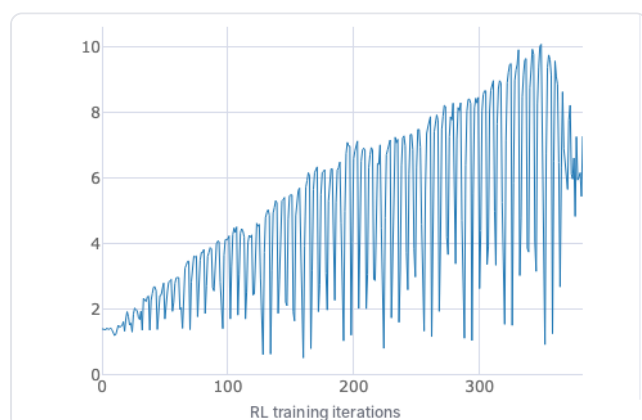
Each entry below summarises one internal review: where the programme stood, what the team worked on, and what it found. Figures are taken from the review slides, with dashboard legends replaced where needed. Most come from deliberately simplified settings, often a single task or a single controlled hyperparameter, used to study one question at a time.

22 Sep 2025 Finding a training set-up that works

Where we were. The aim was to settle the reinforcement-learning configuration for a first Paramorph MVP: the training settings, the observations and actions, and an efficient compute set-up. The immediate problem was that learning failed even on a simple task. On OLMo 2 tiny (about 5M parameters, two transformer layers), Paramorph finished with a validation loss of 5.59 against 4.88 for a tuned baseline, and the randomness of its policy grew during training instead of shrinking (Figure B1a).

What we did. The team set out controlled experiments on each candidate cause: the RL algorithm's settings, tested first in a simple environment; the effect of clipping hyperparameters to their allowed range; alternative action designs; observation ablations, including the feature set from earlier published work on learned hyperparameter control (Almeida et al., 2021); the size of the policy network; and independent versus centralised critics. In parallel it measured training throughput on L40S, A100 and H100 GPUs.

What we found. Throughput scales with the number of environment runners executing in parallel, not with the number of environments queued within each runner (Figure B1b), which set how later training jobs were laid out.



— Learning-rate agent: policy entropy

Figure B1a. The first problem, September 2025. The randomness of the learning-rate agent's policy grew steadily during training instead of shrinking, a sign that it was not converging. Diagnosing this, including a bug in the open-source RL library, occupied the first two months.

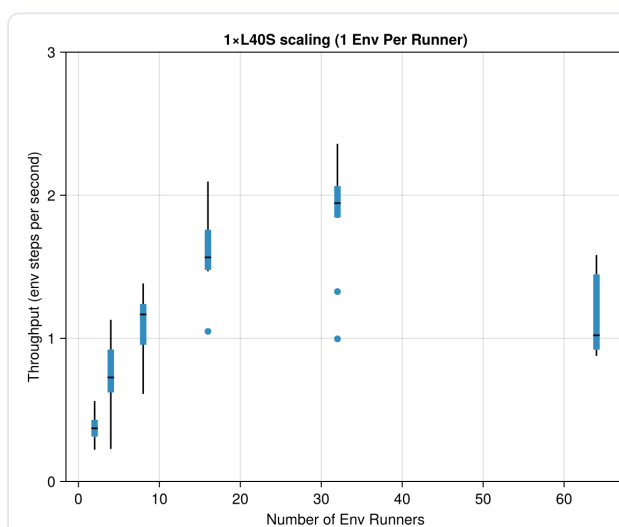


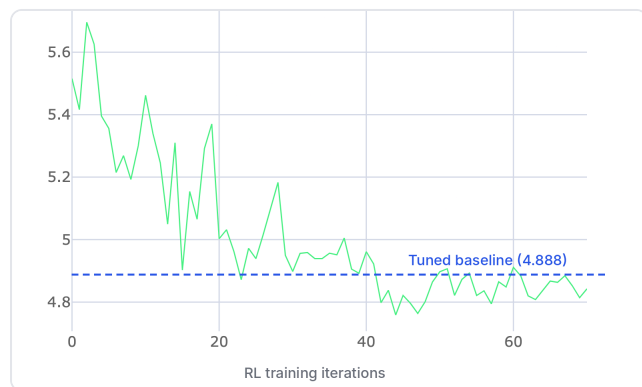
Figure B1b. Capacity testing, September 2025. Environment throughput on a single L40S GPU as the number of parallel environment runners grows, with small MLP inner tasks. Measurements of this kind set how many training runs each Paramorph training job runs in parallel.

3 Nov 2025 First stable learning

Where we were. The previous sprint had found and fixed an important bug in the open-source RL library, but stable learning on a single task had still not been achieved.

What we did. The team focused on that one task. The decisive change was making the amount of experience collected per update comparable to the length of a training run.

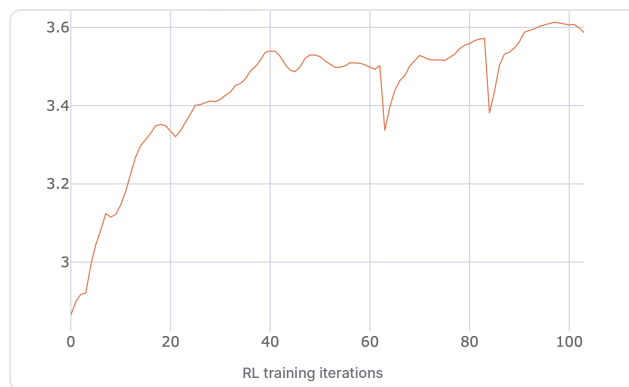
What we found. For the first time, Paramorph learned clearly and stably, reaching the tuned baseline on OLMo 2 tiny (Figure B2). Further experiments showed that training only on short runs does not generalise to longer ones, so short runs belong in the training mix but cannot replace long ones. Results on BERT were solid; poor results on GPT-2 were traced to a bug, since fixed. On MLP tasks, a training instability appeared only when gradient-based observations were enabled, coinciding with some layers' gradients collapsing to zero, which pointed to an interaction with the inner model's activation function.



— Paramorph: mean final validation loss

--- Tuned baseline

Figure B2a. First stable learning, November 2025. OLMo 2 tiny (about 5M parameters). Two months earlier the same setup ended at 5.59.



— Mean episode return

Figure B2b. The same run, seen through the training signal. Mean return rises steadily as the policy learns. The two sharp dips are a logging artefact from a test of recovery from checkpoints.

14 Jan 2026 Regrouping after the first MVP attempt

Where we were. A first MVP training run had been launched in December. Rather than push on with it, the team stepped back to strengthen the base set-up.

What we did. It corrected how parameter-update observations were computed, moved inner-model training from fp16 to bf16, changed how training sequences are sampled and added further observations. It also attacked training speed: removing redundant loss computation (about 13% faster on Hugging Face tasks), packing sequences instead of padding them (10% to 40%) and using smaller tokenisers (50% to 200%). Finally, it began building an “experiment funnel” of cheaper intermediate settings between the simplest tests and full-scale runs.

What we found. A new run with ten times fewer parallel environments already showed a steadier rise in reward.

17 Feb 2026 Many fast experiments

Where we were. Large-scale experiments were too expensive and noisy to settle design questions with confidence.

What we did. With roughly a 2× speed-up in hand, the experiment funnel came into use: MLP tasks, “nano” language models and a small OLMo 3 task, with MuJoCo and a simplified re-implementation of the environment available for even cheaper tests. Working mostly in a single-task, learning-rate-only setting, the team ran systematic ablations covering the number of policy-update epochs, advantage normalisation, how the policy’s exploration noise is parameterised, activation functions, the shape of the policy network and alternative action dynamics. An OLMo 3 trainer was integrated.

What we found. Several ablations measurably stabilised training and became defaults. With all hyperparameter agents switched on, the nano-model policy learned clearly but did not yet reach the tuned baseline.

3 Mar 2026 Rethinking the training signal

Where we were. Analysis suggested that the training signal then in use was hard to optimise, encouraging agents to over-focus on the early part of a run.

What we did. The team simplified further, to one task, one hyperparameter and a single global agent, and tested several alternative formulations. It introduced two evaluation measures used throughout the rest of the programme: the number of steps to reach a target loss, and the share of runs that reach it at all. It also ran a dedicated bug-finding effort.

What we found. A hand-specified formulation produced stable, interpretable learning on the single task but could not scale to a distribution of tasks, which set the requirement for the next stage.

20 Mar 2026 From one task to many

Where we were. The team needed a training signal that would work across tasks without hand-tuning.

What we did. It designed and tested two new formulations that generalise across tasks, then trained on two tasks at once and on a family of GPT-2 tasks. It also ran first experiments with per-layer control and with six controlled hyperparameters.

What we found. Both new formulations performed comparably. Paramorph trained stably on two tasks with different architectures and very different datasets at once, and on the GPT-2 family it produced consistent, task-adapted schedules (Figure B3), with early signs of generalisation across dataset, family and size. Per-layer control learned more slowly than global control but reached better final performance, and its schedules were interpretable (Figure B4). Six-hyperparameter control learned clearly and outperformed learning-rate-only control, though it did not yet reach the tuned baseline.

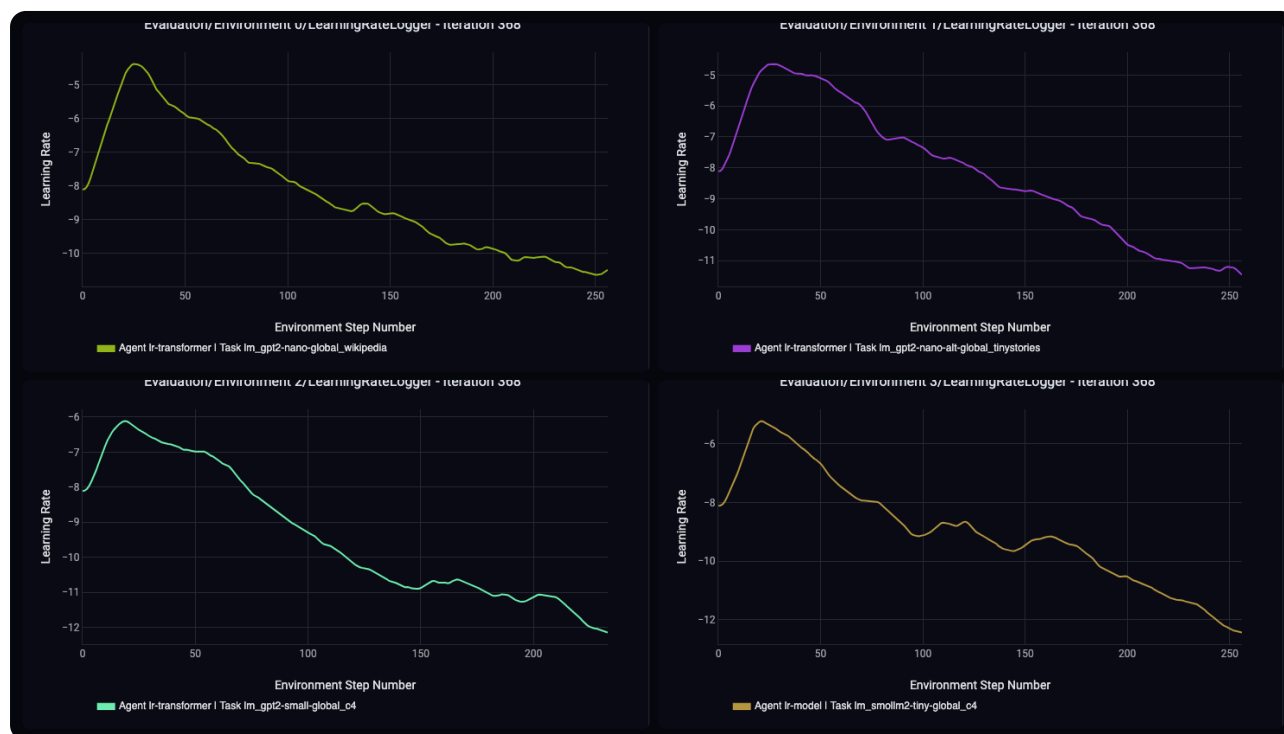


Figure B3. One policy, four tasks, March 2026. Learning-rate schedules (log scale) chosen by a single policy trained across a family of GPT-2 tasks, on four evaluation tasks: **gpt2-nano** on Wikipedia and on TinyStories, **gpt2-small** on C4 and SmoLLM2 tiny on C4. The policy produces a consistent warm-up and decay, adapted in timing and level to each task.

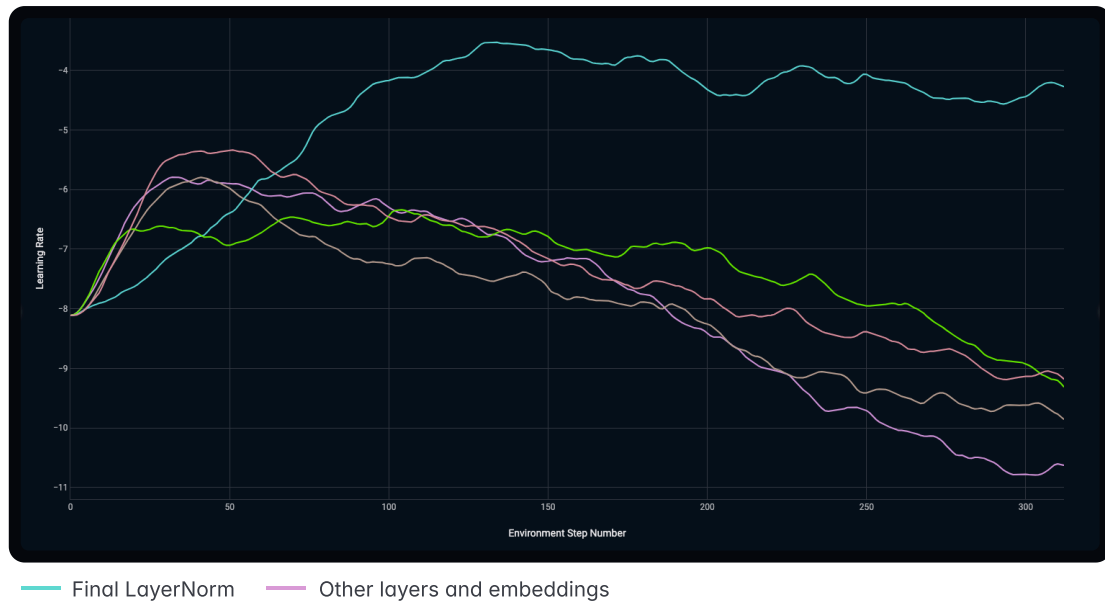


Figure B4. Per-layer control, March 2026. Learning rate (log scale) chosen for each layer of `gpt2-nano`. Every layer receives a warm-up and decay except the final LayerNorm, which the policy treats differently.

21 Apr 2026 The first GPT-2 foundation-model run

Where we were. With a training signal that generalised, the team could train a single policy across a distribution of GPT-2 models of different sizes, with global six-hyperparameter control, on B200 GPUs for five days.

What we found. Speed-ups of roughly 2× to 10× over Hugging Face defaults, and 1.5× to 3× over the previous MVP. The policy began to beat tuned baselines on some tasks, learning was still improving when the review took place (Figure B5), and results generalised across model family, dataset, batch size and context length. Generalisation across model size was partial, and the official 124M GPT-2 was the weakest case.

Next. Improve scaling to larger models, prepare for external testing, launch a further large run and begin OLMo 3 training.



Figure B5. The first GPT-2 foundation-model run, April 2026. Speed-up against each baseline at successive Paramorph checkpoints (horizontal axis: RL training iterations), on a held-out task and two variants of it. Gains against the untuned baselines grow as training proceeds, and the policy begins to match the tuned baselines at some checkpoints.

12 May 2026 Tuning and scaling out

What we did. The team tuned the RL set-up itself (policy-network size and learning rate) and tested whether published learning-rate scaling laws could set Paramorph's starting point automatically. Using special credits on AWS HyperPod, it ran the foundation-model training on one and on five B200 nodes.

What we found. Policy networks of different sizes performed comparably. The scaling-law starting points caused instabilities at the smallest model sizes, where the laws do not hold well, and published results suggest only about a tenfold difference in starting learning rate between 1M and 1B parameters, a range Paramorph can already cover. The five-node run performed better per RL step, and both performed at least as well as the April run. On the 124M GPT-2, even the tuned baselines offered only about 2× over Paramorph's constant starting configuration, leaving less headroom on that task.

9 Jun 2026 Diagnosing the largest model

Where we were. The 124M GPT-2 remained the hard case. A policy trained on that single model achieved only about 15% speed-up over the constant starting configuration and none over the tuned baselines.

What we did. The team began a diagnostic programme: further single-model runs, a simplified learning-rate-only setting, ablations on smaller models to locate where learning broke down, and a test of the initial learning rate. It revisited scaling laws with a rule that also accounts for batch size, and ran the OLMo 3 model ladder from 1M to 1B parameters using AllenAI's official configurations.

What we found. Metrana's synchronised cursor revealed that sharp drops in loss coincided with the Adam β_2 control reaching its ceiling (Figure B6), suggesting β_2 mattered more than assumed. On the OLMo 3 ladder, learning was evident mainly at the smaller, in-distribution sizes. A single-model run on OLMo 3 at 30M parameters, using AllenAI's own settings including a Chinchilla-optimal batch size, was estimated by inspection at about 1× against the OLMo recipe at its standard training length and about 3× at six times that length.

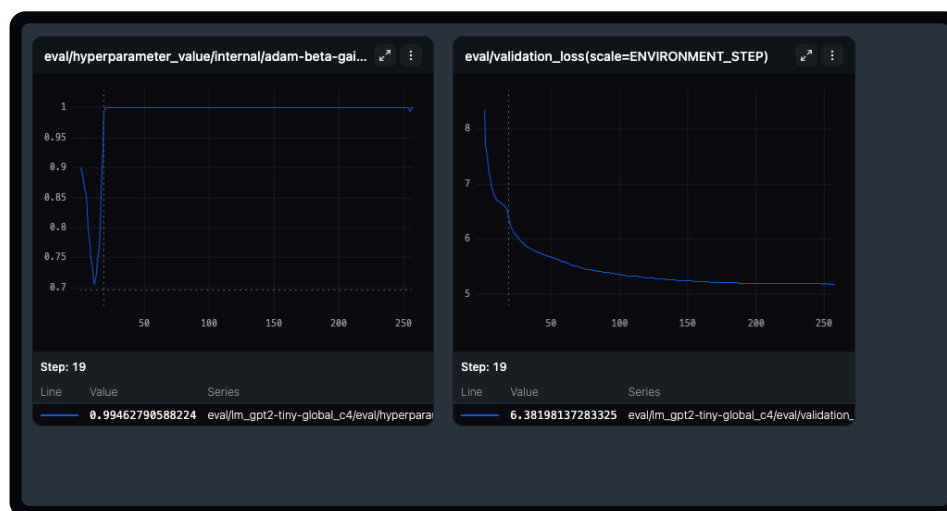


Figure B6. A discovery made in Metrana, June 2026. Left: the Adam β_2 control on `gpt2-tiny`. Right: validation loss for the same run. Metrana's synchronised cursor (dotted line) showed that the sharp drop in loss coincided with β_2 reaching its ceiling, which led to a redesign of how the Adam momentum terms are controlled.

22 Jul 2026 Short-sightedness found and fixed

Where we were. On the 124M GPT-2, Paramorph made strong early progress and then plateaued (Figure B7a).

What we did. The team simplified to a single model, a fixed training length and learning-rate-only control, and tested one hypothesis at a time: a lower initial learning rate (no effect), fixed-length training alone (no effect), early stopping rules (mixed), reward shaping towards later targets (no clear effect) and longer rollouts (clearly

helpful but not sufficient).

What we found. Analysis of how cumulative reward built up within runs, which Metrana was already logging, showed that the RL algorithm's effective planning horizon made short-sighted policies optimal. A change to the training configuration removed the effect. On `gpt2-nano`, six-hyperparameter control then consistently beat the fully tuned baseline (Figure B8), and on the 124M GPT-2 a single-model policy began to learn clearly (Figure B7b). A study of training length showed that policies trained on runs of varying length learned one compromise schedule rather than adapting to each length (Figure B9a). Per-layer control was now less stable than global control, which the team attributed to multi-agent training details still to be resolved.

Next. External testing, scaling to much larger batch sizes and much longer runs, choosing which hyperparameters to control, and planning how to scale RL training itself.

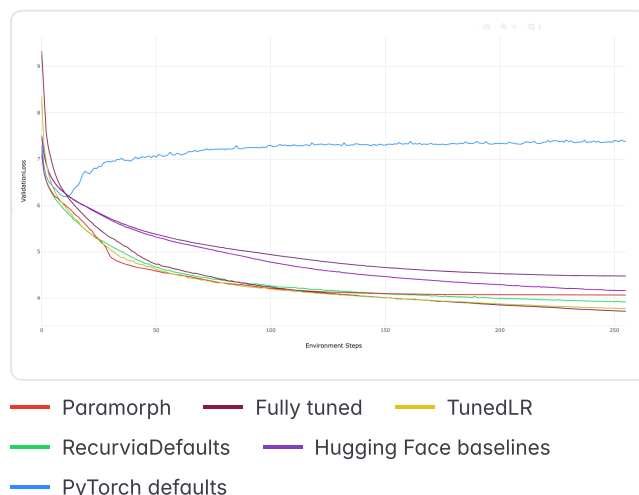


Figure B7a. The plateau, before July 2026. Validation loss on the official 124M GPT-2. Paramorph leads early, then flattens and finishes behind the tuned baselines and its own constant starting configuration.

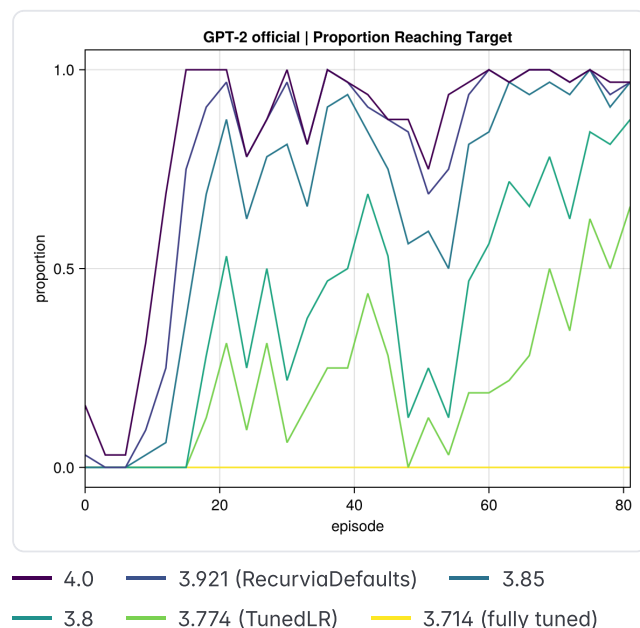


Figure B7b. After the fix, July 2026. Share of evaluation runs reaching each target validation loss on the official 124M GPT-2 (single-model policy, horizontal axis: RL training episodes). The share reaching the learning-rate-tuned baseline's loss rises to about two-thirds; the fully tuned baseline's loss was not yet reached.



Figure B8. After the July 2026 fix, on `gpt2-nano`. Share of evaluation runs that reach each target validation loss, as RL training progresses. All three targets are below the fully tuned baseline's loss of about 3.22. With six-hyperparameter control, almost every run reaches 3.20 by the end of training, and harder targets are reached increasingly often. Learning-rate-only control reaches none of them. These policies were trained on this single task.

1 Sep 2026 Preparing the foundation-model run

What we did. The team settled the configuration for a large run on 13 September: global control, an improved way of controlling the Adam momentum terms prompted by the June β_2 observation, and changes to help the policy adapt to different training lengths. It continued testing learning-rate scaling rules based on batch size and model size, and removed unused code to make the codebase easier for new researchers.

What we found. Giving the policy its training length as an input and reducing early exploration produced schedules that genuinely depend on the budget (Figure B9b), though not yet fully. Results from the scaling rules were mixed. The run launched on 13 September is the one evaluated in Sections 4 to 7 of this report.

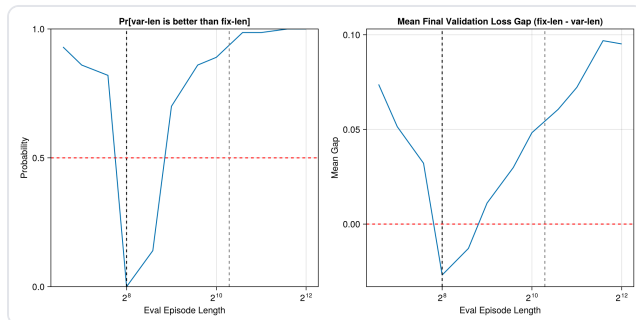


Figure B9a. Studying training length, July 2026. Policies trained on variable-length runs compared with policies trained at a single fixed length (256 steps, black dashed line), evaluated at a range of lengths up to 2^{12} steps. Left: probability that the variable-length policy finishes with the lower loss. Right: the mean gap in final loss. The fixed-length policy wins only near its own training length.

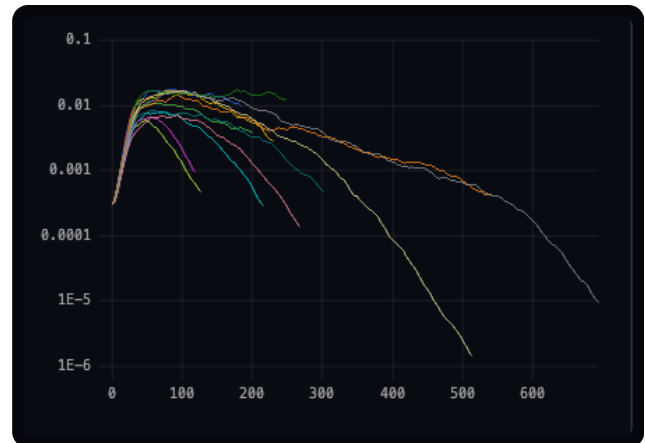


Figure B9b. Budget-dependent schedules, September 2026. Learning rate (log scale) chosen by the policy for runs of different lengths, after the training budget was added as an observation. Each line is a different budget; shorter runs decay earlier and faster.

C Appendix: per-task results

Speed-up for every task in the evaluation suite against each baseline, averaged over the seeds, of five, that reached the baseline's best loss. "none" means Paramorph did not reach the baseline's best loss within the budget on any seed; "—" means the baseline was not run for that task. OLMo 3 task names omit the common suffix `_olmo-mix-1124_1024-squeeze`. The underlying runs can be inspected in Metrana ([multiseed-eval runs](#)). Column headings abbreviate the baselines defined in Section 4.2.

Task	HF	HF const. LR	Recurvia	Recurvia scaled	PyTorch	TunedLR	TunedFull
gpt2-nano_wikipedia	22.7×	10.4×	3.6×	3.4×	3.0×	2.2×	none
gpt2-tiny_c4	20.8×	9.4×	3.9×	3.8×	2.2×	2.1×	none
gpt2-small_c4	5.8×	3.8×	2.2×	5.7×	1.6×	1.6×	none
gpt2-off_c4	3.9×	2.7×	1.7×	6.1×	1.0×	1.3×	1.0×
smollm2-tiny_c4	39.0×	67.0×	17.6×	10.6×	4.2×	9.7×	2.1×
smollm2-small_c4	11.5×	43.6×	2.2×	40.4×	1.9×	—	—
qwen3-tiny_c4	55.0×	55.0×	257×	7.7×	15.2×	—	—
qwen3-small_c4	9.6×	14.2×	5.3×	49.7×	3.6×	—	—
qwen3-base_c4	20.5×	19.8×	1.9×	39.2×	1.8×	—	—
olmo2-tiny_c4	9.5×	9.5×	4.0×	3.8×	2.2×	3.3×	1.6×
olmo3-14M	196×	127×	12.9×	11.7×	7.4×	—	—
olmo3-30M	196×	196×	4.3×	15.5×	3.6×	—	—
olmo3-60M	45.6×	45.6×	3.2×	39.0×	2.7×	—	—
olmo3-100M	26.1×	26.1×	2.4×	51.0×	2.2×	—	—
olmo3-370M	15.1×	15.1×	1.8×	38.7×	1.9×	—	—
olmo3-600M	10.0×	10.0×	1.4×	50.4×	1.5×	—	—
olmo3-760M	9.1×	9.1×	1.2×	37.2×	1.5×	—	—
olmo3-1B	6.9×	6.9×	1.2×	39.6×	1.4×	—	—
gpt2-nano_wikipedia_len16	17.0×	8.5×	2.1×	1.9×	none	—	—
gpt2-nano_wikipedia_len32	16.5×	7.9×	2.2×	2.1×	1.1×	—	—
gpt2-nano_wikipedia_len64	16.3×	9.3×	2.1×	2.0×	1.2×	—	—
gpt2-nano_wikipedia_len128	18.0×	11.3×	2.2×	2.2×	1.8×	—	—
gpt2-nano_wikipedia_len256	23.4×	10.9×	3.6×	3.5×	2.7×	—	—
gpt2-nano_wikipedia_len512	21.3×	9.3×	4.8×	4.8×	3.9×	—	—
gpt2-nano_wikipedia_len1024	17.7×	11.8×	7.4×	7.3×	6.7×	—	—
gpt2-nano_wikipedia_len2048	25.2×	18.1×	13.2×	13.1×	12.3×	—	—
gpt2-nano_wikipedia_len4096	35.4×	27.7×	24.8×	24.6×	21.9×	—	—
gpt2-nano_wikipedia_batch8	15.3×	7.5×	2.2×	2.7×	2.2×	—	—
gpt2-nano_wikipedia_batch16	18.7×	9.9×	2.8×	3.1×	2.5×	—	—
gpt2-nano_wikipedia_batch32	23.4×	12.0×	3.4×	3.3×	2.8×	—	—
gpt2-nano_wikipedia_batch64	29.3×	13.8×	4.3×	3.8×	3.4×	—	—
gpt2-nano_wikipedia_batch128	36.4×	18.4×	6.0×	4.4×	4.1×	—	—
gpt2-nano_wikipedia_batch256	42.7×	23.8×	8.4×	5.3×	4.8×	—	—
gpt2-nano_wikipedia_batch512	51.2×	28.6×	8.6×	5.3×	5.5×	—	—
gpt2-nano_wikipedia_batch1024	63.8×	35.8×	11.4×	5.8×	6.2×	—	—
gpt2-nano_wikipedia_batch2048	64.0×	42.8×	15.7×	7.7×	8.4×	—	—
gpt2-nano_wikipedia_ctx128	64.3×	14.4×	4.8×	4.2×	3.4×	—	—
gpt2-nano_wikipedia_ctx256	85.7×	16.3×	7.5×	4.2×	4.1×	—	—
gpt2-nano_wikipedia_ctx512	120×	16.7×	7.0×	5.4×	4.0×	—	—
gpt2-nano_wikipedia_ctx1024	128×	16.1×	7.2×	3.8×	4.6×	—	—
gpt2-tiny_c4_ctx256	24.8×	12.9×	4.3×	3.9×	2.2×	1.7×	none