Overcoming Forgetting in LLM Fine-Tuning with Evolution Strategies

Kajetan Schweighofer
Cognizant AI Lab
kai.schweighofer@gmx.at

Conor F. Hayes
Cognizant AI Lab
conor.hayes@cognizant.com

Roberto Dailey
Cognizant AI Lab
roberto.dailey@cognizant.com

Risto Miikkulainen
UT Austin & Cognizant AI Lab
risto@cs.utexas.edu

Xin Qiu
Cognizant AI Lab
qiuxin.nju@gmail.com

Abstract

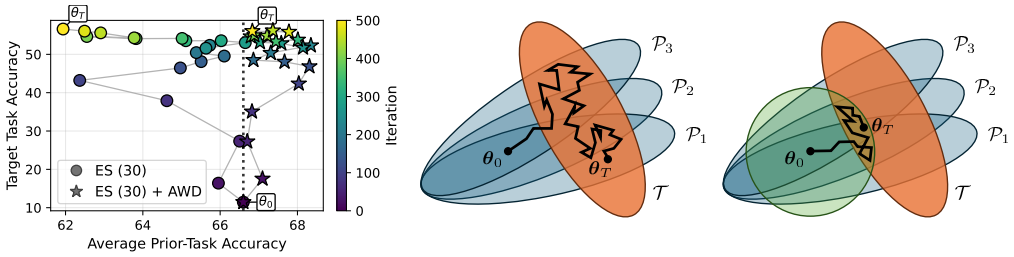
Evolution Strategies (ES) has recently emerged as a competitive alternative to reinforcement learning (RL) for large language model (LLM) fine-tuning, offering advantages through simplicity, scalability, and inference-only training. However, recent work suggests that ES fine-tuning on new tasks may induce forgetting of prior tasks. First, this paper shows that prior task forgetting (1) is better characterized as performance drift rather than irreversible forgetting, with prior-task performance often recovering during ES training; and (2) is not a specific failure mode of ES, but can also arise for fine-tuning with RL methods. Second, it analyzes when and why such drift arises, highlighting its dependence on ES training dynamics, particularly random walk behavior in weakly constrained directions of the weight space. Third, based on these insights, it introduces *Anchored Weight Decay* (AWD) as a parameter-space regularization technique that constrains optimization toward the initial model parameters. AWD effectively stabilizes prior-task performance while preserving target-task performance, achieving benefits comparable to large ES population sizes at much lower computational cost. Thus, contrary to previous beliefs, the paper shows that prior-task forgetting under ES is largely avoidable, positioning ES as a promising approach for continual learning in LLMs.

1 Introduction

Ever-increasing amounts of compute are spent on post-training of LLMs, adapting them to particular tasks through reward signals [Cobbe et al., 2021, DeepSeek-AI, 2025]. Reinforcement Learning (RL), in particular Group Relative Policy Optimization (GRPO) [Shao et al., 2024], has become the standard approach in this setting. However, RL-based training is complex with high engineering effort and limited scalability. Recently, Evolution Strategies (ES) have reemerged as a promising alternative for LLM post-training [Salimans et al., 2017, Qiu et al., 2025]. ES optimizes model parameters through stochastic perturbations in weight space, requiring only forward passes. This makes ES particularly attractive for large-scale deployment, as it can leverage optimized inference systems and parallelize naturally across population members.

The code to reproduce all experiments is available at https://github.com/kschweig/es-awd

Preprint.



(a) AWD mitigates forgetting (b) Standard ES optimization (c) ES with AWD

Figure 1: *Anchored Weight Decay (AWD) mitigates prior task forgetting.* (a) Target task accuracy (Countdown) vs. average prior task accuracy. The dotted line denotes prior task accuracy for the original model with weights $\boldsymbol{\theta}_0$. Standard ES shows unstable prior-task accuracy during training iterations (denoted by color), while AWD stabilizes it. (b) Prior tasks $\mathcal{P}_i$ and target task $\mathcal{T}$ occupy different high-performance regions in weight space. ES exploration within $\mathcal{T}$ can cause drift on prior tasks. (c) AWD constrains optimization near the initial model, mitigating forgetting.

While ES has been shown to be competitive with, and in some cases outperform, RL-based fine-tuning [Qiu et al., 2025, Hoy et al., 2026, Gan and Isola, 2026], recent work reports that ES induces forgetting of previously learned tasks [Abdi et al., 2026, Hoy et al., 2026]. These results raise concerns about the suitability of ES for continual learning scenarios, where preserving prior capabilities is essential. However, despite demonstrating this effect, the nature, causes, and generality of the phenomenon remain insufficiently understood. Key open questions include whether it reflects irreversible loss of knowledge or a transient effect, whether similar forgetting can arise in RL methods, how it depends on the target task and model family, and whether the larger magnitude of weight updates in ES are indeed the primary cause (as suggested by Abdi et al. [2026]). Moreover, given that forgetting arises in at least some settings, an important next step is to develop effective mitigation strategies.

This paper provides a comprehensive analysis of prior-task forgetting under ES fine-tuning over a diverse set of target tasks, prior tasks, model families and model sizes. It shows that forgetting is better characterized as *performance drift* rather than irreversible forgetting, as prior-task performance often recovers during training. The experiments further show that forgetting also occurs with RL methods, indicating that forgetting is a broader issue in LLM fine-tuning rather than one specific to ES. For ES, its training dynamics are experimentally identified as the main driver of forgetting, in particular, a random walk in directions of weight space that are weakly constrained by the target task. Building on this insight, the paper introduces *Anchored Weight Decay (AWD)*, a simple modification to the ES update that constrains optimization toward the initial model parameters. As illustrated in Fig. 1, AWD reduces task-irrelevant drift while retaining the benefits of ES, stabilizing prior-task performance without sacrificing target-task accuracy.

2 Background and Related Work

Two methods for fine-tuning LLMs are considered in this work: ES, a population-based, gradient-free approach that samples in weight space, and GRPO, a policy-gradient RL method that samples in token space. An introduction to ES is given in Sec. 2.1, to GRPO in App. B. Moreover, a discussion of continual learning and prior-task forgetting, particularly in LLMs, is provided in Sec. 2.2.

2.1 Evolution Strategies

Evolution Strategies (ES) are a class of population-based, zeroth-order optimization methods that update model parameters via stochastic perturbations and reward aggregation, rather than gradient-based backpropagation. The objective of ES is to choose weights $\boldsymbol{\theta}$ that maximize the expected reward under Gaussian perturbations: $\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},I)}\left[R(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon})\right]$. Given model parameters $\boldsymbol{\theta}$ and a reward function $R(\cdot)$, ES samples perturbations $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, I)$, scales them by σ, and evaluates reward of perturbed models $r_n = R(\boldsymbol{\theta} + \sigma\boldsymbol{\epsilon}_n)$. Subsequently, rewards are normalized, for example via z-scaling over the population $\hat{r}_n = \frac{r_n - \mu_N}{\sigma_N}$, where μ_N and σ_N are the mean and standard deviation over the population [Qiu et al., 2025]. The weight update then aggregates these perturbations weighted

by their normalized rewards $\hat{r}_n$, through the following update rule:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \alpha \cdot \frac{1}{N} \sum_{n=1}^{N} \hat{r}_n \boldsymbol{\epsilon}_n \ , \tag{1}$$

where α is the stepsize of the update. The full algorithm is provided in Algorithm 1. Eq. (1) can be interpreted as a Monte Carlo estimate of the gradient of an objective that is Gaussian-smoothed in weight space [Qiu et al., 2025]. Classical ES methods date back to Rechenberg [1973] and Schwefel [1977], with modern variants such as NES [Wierstra et al., 2008, 2014] and CMA-ES [Hansen and Ostermeier, 2001] improving search efficiency. Further, Salimans et al. [2017] demonstrated that ES can scale to deep neural networks and achieve competitive performance with RL in control tasks.

Recently, ES has been revisited as an alternative paradigm for fine-tuning LLMs [Qiu et al., 2025, Korotyshova et al., 2025, Sarkar et al., 2025], particularly in settings where only outcome-level rewards are available. Unlike RL methods that explore in token space, ES performs exploration in weight space, allowing it to naturally handle long-horizon and sparse reward signals while avoiding token-level credit assignment. Surprisingly, fine-tuning billion-parameter models is possible with very small population sizes; this phenomenon has been linked to low-dimensional curvature of fine-tuning landscapes [Liang et al., 2026]. Moreover, ES relies solely on forward passes thus can utilize specialized hardware and software for inference and is easily parallelized.

2.2 Continual learning and prior-task forgetting

Catastrophic forgetting was first identified as an issue by McCloskey and Cohen [1989], who showed that sequential training leads to rapid loss of previously learned knowledge in neural networks. To address this, replay-based methods [Lopez-Paz and Ranzato, 2017, d'Autume et al., 2019] store and reuse past examples during training on new tasks. In contrast, regularization-based approaches [Kirkpatrick et al., 2017, Zenke et al., 2017] constrain changes to model parameters important for prior tasks. These ideas are closely related to the proposed method AWD as discussed in App. C. Another line of work freezes previously learned parameters and expands the model architecture with additional parameters that are trained on new tasks [Rusu et al., 2016]. A comprehensive survey of continual learning and prior-task forgetting is given by Parisi et al. [2019].

Recent work has examined prior-task forgetting in the context of LLMs. Scialom et al. [2022] showed that sequential instruction tuning leads to performance degradation across tasks such as summarization and style transfer, highlighting instability in cross-task generalization. Complementing this, Luo et al. [2025] provided a large-scale empirical study demonstrating that catastrophic forgetting consistently occurs during continual fine-tuning of LLMs, affecting domain knowledge, reasoning, and comprehension, and even increasing with model scale. To mitigate these effects, Wang et al. [2023] proposed O-LoRA, which learns task-specific updates in orthogonal low-rank subspaces to reduce interference between tasks. Wu et al. [2024] adopted an architectural strategy that expands transformer blocks for new tasks while freezing existing parameters, increasing model capacity to preserve prior knowledge. Shenfeld et al. [2025], Chen et al. [2025], Lai et al. [2026] investigated forgetting of prior tasks under RL and supervised fine-tuning, finding that RL methods lead to less forgetting compared to supervised fine-tuning.

Most closely related to this paper are the recent studies by Abdi et al. [2026] and Hoy et al. [2026] on prior-task forgetting under ES fine-tuning. Abdi et al. [2026] demonstrate that ES fine-tuning on a target task can lead to prior-task forgetting, attributing it to larger weight updates compared to GRPO. This paper extends their setting by systematically investigating different target tasks, prior tasks and model architectures, as well as the influence of ES population size, thus clarifying the reason for larger magnitudes of weight updates, and ultimately introducing AWD as mitigation strategy for forgetting. Hoy et al. [2026] investigated forgetting of ES vs. GRPO in a sequential learning setting on multiple tasks. They provided an analysis of the geometry of weight changes, showing that GRPO and ES solutions are linearly mode connected though their update directions per iteration are near-orthogonal. Furthermore, they suggest a theoretical model of optimization with ES, showing that ES induces a random walk in directions of the weight space that are weakly coupled to the target task. To mitigate forgetting, they suggest early stopping, balancing prior task forgetting with performance on the target task. This paper further investigates the relationship between the random walk behavior of ES and prior-task forgetting, motivating AWD as a simple yet effective method to mitigate forgetting.

3 Experimental Setup

All experiments in this paper follow a common setup designed to study prior-task forgetting. This setup includes multiple complementary prior tasks and three diverse target tasks for training.

Target tasks. Three different target tasks are considered, Countdown [Gandhi et al., 2024, Pan et al., 2025], GSM8K [Cobbe et al., 2021] and ProofWriter [Tafjord et al., 2021]. These tasks focus on arithmetic reasoning, multi-step numerical reasoning, and logical reasoning, respectively. Training was performed for a fixed set of 200 examples.

Prior tasks. As prior tasks, HellaSwag [Zellers et al., 2019], PIQA [Bisk et al., 2020], ARC-Challenge [Clark et al., 2018] and MMLU-Pro [Wang et al., 2024] are considered. They focus on selecting plausible sentence continuations, commonsense reasoning, and professional and academic knowledge across multiple disciplines. Furthermore, target tasks not trained on within an experiment are considered as additional prior tasks.

Evaluation. Evaluation was conducted on 500–2,000 examples per task, depending on how many were available; for target tasks, evaluation sets were disjoint from the training data. When more than 2,000 examples were available, a random subset of 2,000 was sampled. Greedy decoding is used for evaluation and accuracy reported.

Model. In line with Abdi et al. [2026], who first identified forgetting through ES fine-tuning, the Qwen-2.5 3B Instruct model [Qwen et al., 2025] is used, unless stated otherwise. The chat template is applied, and the maximum generated sequence length capped at 1024 tokens, which is sufficient to generate valid reasoning and answers for all tasks considered.

Implementations of ES and GRPO. The ES implementation follows Qiu et al. [2025], using their optimized variant leveraging vLLM [Kwon et al., 2023] for efficient sequence generation and scoring of individual members. For GRPO, the implementation provided through the `verl` framework [Sheng et al., 2024] was used. This setup is consistent with Abdi et al. [2026], who used the same implementations. ES is used with population size 30 if not otherwise specified, GRPO is used with KL penalty. Additional details on the experimental setup are provided in App. D.

4 Understanding Forgetting in LLM Fine-Tuning

To motivate a mitigation strategy for prior-task forgetting in LLM fine-tuning with ES, it is first examined when and why such forgetting arises. Specifically, the forgetting behavior of ES and GRPO is compared throughout the training process. Furthermore, the dependence of forgetting on the target task, prior task and model types is examined. Finally, the role of ES training dynamics in driving forgetting is investigated, suggesting a mitigation strategy.

4.1 Forgetting throughout the training process

First, the change in prior task accuracy over the training process is investigated. Exemplary results for ES and GRPO under the standard experimental conditions (Sec. 3) for the three different target tasks

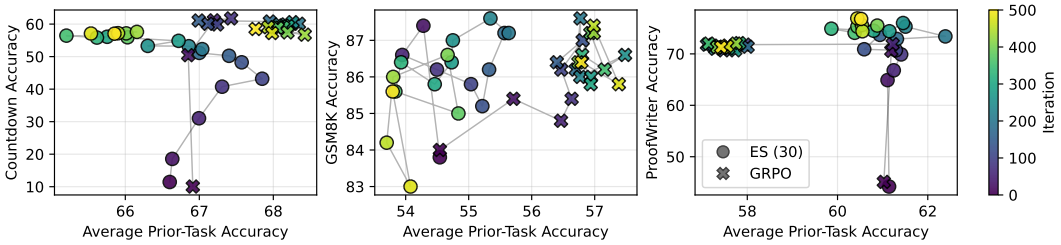


Figure 2: *Target task accuracy vs. average prior task accuracy throughout training.* Colors denote the iteration within the training process. For ES, prior task accuracy exhibits noticeable drift, where accuracy decreases initially, but often recovers later in training. Moreover, forgetting does not arise under all settings. GRPO exhibits less drift in prior task accuracy, but can also lead to severe forgetting in some settings. Detailed results per prior-task are provided in Fig. 3, Fig. 8 and Fig. 9.

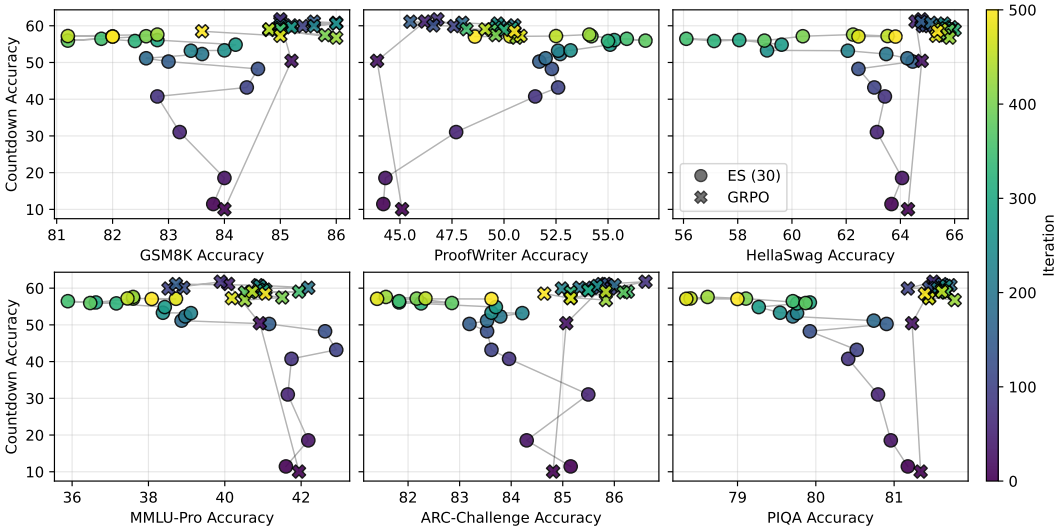


Figure 3: *Individual prior task accuracies training on Countdown as target task.* Colors denote the iteration within the training process. A performance drift rather than irreversible forgetting is observed across multiple tasks. For example, performance on HellaSwag drops by 8% accuracy over the first 300 iterations, but subsequently recovers to its original level by the final iteration.

Countdown, GSM8K and ProofWriter are provided in Fig. 2. Individual results per prior-task for the Countdown experiment are provided in Fig. 3. For GSM8K and ProofWriter, they are provided in Fig. 8 and Fig. 9 in App. E.

The results for Countdown as target task are largely consistent with those reported by Abdi et al. [2026]: ES exhibits prior-task forgetting, whereas GRPO does not. However, the change in performance for some individual prior tasks (HellaSwag, MMLU-Pro and ARC-Challenge) shows an unexpected pattern. Performance initially drops, but recovers later in training. For ProofWriter as prior task, the opposite trend is observed, an initial improvement followed by a return to baseline performance. Together, these observations suggest that **ES does not induce irreversible forgetting, but rather a performance drift that is transient**. A possible explanation, as also suggested by Hoy et al. [2026], is that ES is free to explore directions in weight space that are weakly constrained by the target task, which may lead to both decreases and increases in performance on unrelated tasks.

The results for GSM8K and ProofWriter in Fig. 2 show a different picture. For those, there is no pronounced prior task forgetting under ES on average over the considered prior tasks. While for some individual prior-tasks, ES still exhibits a moderate amount of forgetting, there are also some without forgetting. Furthermore, for ProofWriter as target task, GRPO exhibits considerable forgetting. Fig. 9 shows, that this is mostly due to a strong accuracy decrease on GSM8K as prior task, yet also others (Countdown, MMLU-Pro) show degradation. Overall, those findings demonstrate that **forgetting is not a specific failure mode of ES, but can also arise for GRPO.**

While analyzing the training behavior throughout the training process gives insights into the mechanisms of prior-task forgetting, it is hard to compare many different experimental conditions. The subsequent analyses consider only the change in accuracies between the base model and the final training iteration, allowing systematic comparisons between settings. Next, the dependence of forgetting during LLM fine-tuning on the target task and model type is analyzed.

4.2 Dependence of target task and model types on forgetting

Prior work on forgetting in ES and GRPO has focused on a limited selection of target and prior tasks, as well as a single model family and scale [Abdi et al., 2026, Hoy et al., 2026], limiting the understanding of how those choices influence this phenomenon. Therefore, the target task (Countdown, ProofWriter), model family (Qwen, Llama) and model size (1.5B, 3B, 7B) are varied systematically and evaluated across multiple prior tasks to assess the extent to which forgetting depends on those factors.

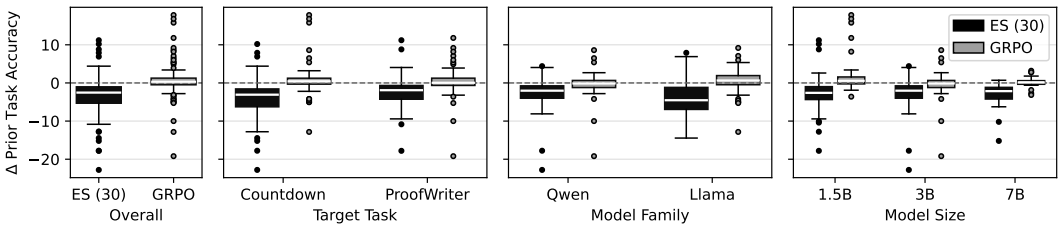


Figure 4: *Change in final prior-task accuracy under different target tasks and base models.* Boxplots for the change (Δ) in prior task accuracy for ES and GRPO. Each boxplot aggregates results across all experimental dimensions (target-tasks, prior-tasks, model types) not defined via the x-axis. Target task, model family (restricting aggregation to 3B models) and model size (restricting aggregation to Qwen models) are varied. No significant dependence on varied factors is observed.

The results are summarized in Fig. 4. ES in the standard setting with a population size of 30 leads to more forgetting than GRPO on average, with a median change in prior accuracy of -2.5 compared to +0.3. However, both methods exhibit strong positive as well as negative changes in accuracy in some settings. Forgetting with both ES and GRPO is comparable under Countdown and ProofWriter as target tasks, with more discrepancy observed between the methods for the Countdown task. Similarly, the discrepancy between ES and GRPO is stronger for Llama models (comparison restricted to the 3B models). Between models of different model sizes (comparison restricted to the Qwen model family), no significant dependence is observed.

Overall, the consistency of observed trends across model families and sizes suggests that the effect is rather a product of the training dynamics of fine-tuning methods than a property of a specific model. Therefore, the impact of ES update characteristics on forgetting is investigated next. As forgetting under fine-tuning with ES appears to be most prominent for the target task Countdown, the remainder of the presented experiments in the main paper focus on this target task.

4.3 The role of ES update characteristics on forgetting

Hoy et al. [2026] argued that ES updates decompose into a signal component and a noise component, the latter inducing a random walk in low-curvature directions that are largely irrelevant to target-task performance. Similar arguments were made by Liang et al. [2026]. Consistent with this view, Qiu et al. [2025] showed empirically that random walk contributes substantially to ES updates. Under the theoretical model for ES updates of Hoy et al. [2026], the expected cumulative weight deviation induced through random walk is given by

$$\mathbb{E}\left[||\boldsymbol{\theta}_T - \boldsymbol{\theta}_0||_2^2\right] = \frac{\alpha^2 T d}{N}\,, \tag{2}$$

where α is the stepsize, T is the number of iterations, d is the number of weights in the model, and N is the population size. Under fixed α and d, it grows linearly in the number of iterations and shrinks inversely with the population size. Thus, less random walk behavior and potentially less forgetting is expected under higher population sizes. This is confirmed by the empirical results (see Tab. 1) showing that larger population sizes substantially reduce forgetting. Furthermore, Sec. 6 empirically validates that Eq. (2) derived by Hoy et al. [2026] for the theoretical model explains the empirical behavior of ES well, as the update norm is indeed inversely proportional to the population size. Overall, the results suggest that random walk in weight space is the primary driver of forgetting in ES fine-tuning. Accordingly, a method to limit the random walk behavior is introduced next.

Table 1: *Change in target and prior task accuracies for different ES population sizes.* The target task is Countdown and the average is calculated over accuracies in all prior tasks, thus excluding Countdown. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. There is less prior task degradation for higher population sizes.

Method	**Countdown**	GSM8K	ProofWriter	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
ES (30)	$+45.3_{(0.3)}$	$-9.5_{(11.5)}$	$-2.0_{(5.7)}$	$-8.2_{(9.0)}$	$-2.8_{(1.0)}$	$-2.0_{(0.5)}$	$-6.5_{(2.6)}$	$-5.2_{(4.7)}$
ES (128)	$+48.6_{(0.9)}$	$+0.9_{(2.0)}$	$-1.4_{(9.3)}$	$+0.5_{(2.0)}$	$-0.9_{(0.3)}$	$-0.4_{(0.4)}$	$-2.4_{(0.7)}$	$-0.6_{(1.9)}$
ES (256)	$+48.4_{(1.1)}$	$-4.2_{(5.7)}$	$+6.5_{(3.7)}$	$+0.1_{(1.7)}$	$-0.7_{(0.6)}$	$-0.1_{(0.5)}$	$-0.0_{(1.7)}$	$+0.3_{(1.8)}$

5 Overcoming Forgetting in ES Fine-Tuning

Given the mechanisms of forgetting outlined in the previous section, an important question is how it can be mitigated. While increasing the population size substantially reduces forgetting, it incurs significant computational cost. This section introduces Anchored Weight Decay, a simple mechanism that achieves comparable improvements even at small population sizes.

Reducing the random walk. The theoretical analysis of ES optimization dynamics by Hoy et al. [2026] suggested that ES updates induce random walk in directions of the weight space that are weakly coupled to the target task. Over multiple iterations, random update directions may accumulate and lead to the prior-task performance drift observed in the experiments. This effect can be counteracted through an explicit constraint that biases the optimization trajectory toward the initial weights $\boldsymbol{\theta}_0$. A principled way to formalize such a constraint is through introducing a regularization term into the ES objective, where $l(\cdot)$ is a penalty function and $\lambda > 0$ controls the strength of the constraint:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},I)} \left[R(\boldsymbol{\theta} + \sigma\boldsymbol{\epsilon})\right] - \lambda \sum_{i=1}^{d} l(\boldsymbol{\theta} - \boldsymbol{\theta}_0)_i \,. \tag{3}$$

This formulation allows for general element-wise penalty functions $l(\cdot)$. For example, $l(x) = \frac{1}{2}x^2$ yields an ℓ_2 penalty inducing smooth shrinkage toward $\boldsymbol{\theta}_0$, while $l(x) = |x|$ corresponds to an ℓ_1 penalty, promoting sparsity in the weight difference $\boldsymbol{\theta} - \boldsymbol{\theta}_0$.

Notably, the objective in Eq. (3) is related to Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017], which considers a quadratic penalty function and weights the regularization of individual parameters by the Fisher information matrix. In contrast, Eq. (3) assigns equal weight to all parameters. This reflects the absence of task-specific importance information in the present setting, as LLMs may be used for a huge variety of prior tasks. Empirical results from prior work [Li et al., 2018] further suggest that such uniform weighting can perform comparably to Fisher-based approaches in practice. A detailed discussion of the relationship to prior work on regularization based mitigation of prior task forgetting is given in App. C.

Anchored Weight Decay. For gradient-based fine-tuning, regularized objectives of the form as in Eq. (3) are usually implemented by adding the penalty term to the loss calculation for backpropagation. That approach is not applicable to ES. However, ℓ_1/ℓ_2 regularization can equivalently be applied as weight decay directly in the update rule of the weights [Hanson and Pratt, 1988, Loshchilov and Hutter, 2019]. Weight decay was introduced with the explicit aim to keep the absolute values of weights close to zero. To limit random walk behavior and thus forgetting, those values instead need to be kept close to the initial weights $\boldsymbol{\theta}_0$. The resulting method is called *Anchored Weight Decay* (AWD). It extends the ES update rule (Eq. (1)) with a decay on the updated parameters:

$$\boldsymbol{\theta}_t = \underbrace{\theta_{t-1} + \alpha \cdot \frac{1}{N}\sum_{n=1}^{N} \hat{r}_n \boldsymbol{\epsilon}_n}_{\text{ES Update (Eq. (1))}} - \alpha\lambda l'\Big(\underbrace{\theta_{t-1} + \alpha \cdot \frac{1}{N}\sum_{n=1}^{N} \hat{r}_n \boldsymbol{\epsilon}_n}_{\text{ES Update (Eq. (1))}} - \boldsymbol{\theta}_0\Big) \,. \tag{4}$$

The full ES algorithm with addition of AWD is provided in Algorithm 1. Whether or not to couple the weight decay term with the learning rate is a design choice that may be investigated in more detail in future work. The ES implementation of Qiu et al. [2025] utilized a fixed α across iterations, thus α could be absorbed into λ. Yet under adaptive α, it may be advantageous to decouple them, as done by Loshchilov and Hutter [2019].

AWD can be understood as a proximal optimization step or a form of trust region method that counteracts the accumulation of task-irrelevant drift in weight space. It retains the simplicity and scalability of ES, requires no additional forward passes, and provides a flexible mechanism to control the geometry of the constraint via the choice of penalty function $l(\cdot)$.

Implementation. Compared to vanilla ES as in Qiu et al. [2025], AWD requires access to the reference parameters $\boldsymbol{\theta}_0$ to compute the decay term. Keeping a copy of $\boldsymbol{\theta}_0$ in VRAM during population evaluation is often impractical, because GPU memory is better utilized for batched sequence generation and reward computation. Instead, $\boldsymbol{\theta}_0$ is stored in contiguous pinned RAM and streamed layer-wise to the GPU on demand during the weight update in Eq. (4). The additional overhead introduced by AWD is negligible in practice, as data transfer is only needed once every iteration and does not interfere with the compute- and bandwidth-intensive evaluation phase.

Algorithm 1 Basic ES Algorithm with Anchored Weight Decay (AWD)

Require: Pretrained LLM with initial parameters $\boldsymbol{\theta}_0$, reward function $R(\cdot)$, total iterations T, population size N, noise scale σ, learning rate α, penalty factor λ, penalty function $l(\cdot)$

for $t = 1$ to T **do** ▷ outer ES iteration over generations of populations
  for $n = 1$ to N **do** ▷ inner ES iteration over population members
    Sample noise $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, I)$
    $r_n \leftarrow R(\boldsymbol{\theta}_{t-1} + \sigma\boldsymbol{\epsilon}_n)$ ▷ evaluate perturbed model on train samples
  end for
  $\hat{r}_n \leftarrow \text{normalize}(r_n)$ ▷ z-scaling rewards over the population
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \alpha \cdot \frac{1}{N}\sum_{n=1}^{N} \hat{r}_n \boldsymbol{\epsilon}_n$ ▷ calculate standard ES update
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t - \alpha\lambda l'(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0)$ ▷ decay towards initial parameters
end for

The decay under ℓ_1 penalty is implemented as proximal step, thus $l'(x) = \min(\alpha\lambda, |x|) \cdot \text{sign}(x)$ instead of $\alpha\lambda \cdot \text{sign}(x)$. However, the proximal step usually leads to better solutions of Eq. (3), as it sets the weight difference exactly to zero (induces sparsity) if it is smaller than $\alpha\lambda$.

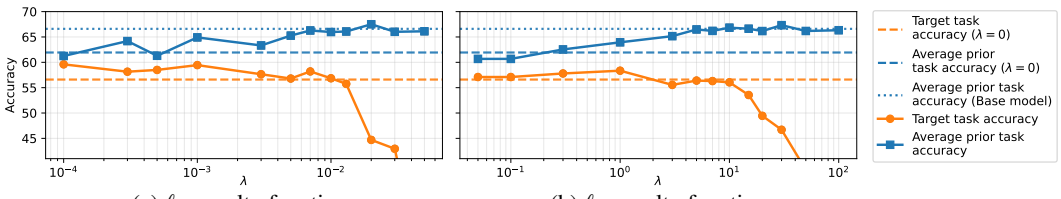
On the choice of penalty function $l(\cdot)$. The effectiveness of applying AWD with ℓ_1 and ℓ_2 penalty to ES is empirically evaluated. Fig. 1 (left) illustrates the difference in optimization behavior between vanilla ES and ES+AWD (ℓ_2), showing that AWD effectively mitigates prior task forgetting. Individual prior-task results are reported in Fig. 10 in the Appendix. A comparison between ℓ_1 and ℓ_2 penalties is provided in Tab. 2; both penalty types were found to lead to similar improvements. Additional results on other combinations of target dataset and model type are provided in App. F, further demonstrating the effectiveness of AWD under both ℓ_1 and ℓ_2 penalties.

On the choice of penalty factor λ. Furthermore, the sensitivity of both target-task performance and prior-task forgetting w.r.t. the penalty coefficient λ is analyzed. As shown in Fig. 5, small values of λ yield little reduction in prior task forgetting compared to ES without AWD. However, target-task performance appears to slightly improve under ℓ_1 regularization at low λ. For both penalty types, there exists a stable range of λ where AWD mostly preserves target performance while substantially reducing forgetting. Beyond this range, prior-task performance remains stable, but target-task performance degrades. The sharp drop at large λ suggests a practical tuning strategy: start with a high λ and decrease it until no performance drop is observed relative to ES without AWD.

AWD was introduced to mitigate forgetting caused by random walk in weight space. While the results confirm its effectiveness, it remains unclear whether this improvement stems from reduced weight drift, which is investigated in the next section.

Table 2: *Change in target and prior task accuracies using AWD with ℓ_1/ℓ_2 decay.* The target task is Countdown and the average is calculated over accuracies in all prior tasks, thus excluding Countdown. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. AWD effectively reduces forgetting for both ℓ_1 ($\lambda = 0.01$) and ℓ_2 ($\lambda = 10.0$) decay.

Run	**Countdown**	GSM8K	ProofWriter	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
ES (30)	$+45.3_{(0.3)}$	$-9.5_{(11.5)}$	$-2.0_{(5.7)}$	$-8.2_{(9.0)}$	$-2.8_{(1.0)}$	$-2.0_{(0.5)}$	$-6.5_{(2.6)}$	$-5.2_{(4.7)}$
ES (30) + AWD (ℓ_1)	$+44.3_{(1.0)}$	$-0.1_{(1.2)}$	$+3.7_{(2.1)}$	$-2.1_{(0.9)}$	$-1.9_{(1.2)}$	$-1.2_{(1.0)}$	$-3.5_{(2.0)}$	$-0.9_{(0.6)}$
ES (30) + AWD (ℓ_2)	$+43.8_{(0.7)}$	$-2.5_{(5.0)}$	$+1.0_{(6.8)}$	$-0.7_{(1.6)}$	$-0.9_{(0.2)}$	$-0.5_{(1.1)}$	$-2.6_{(0.9)}$	$-1.0_{(2.4)}$



(a) ℓ_1 penalty function (b) ℓ_2 penalty function

Figure 5: *Effect of the AWD penalty factor λ on target- and prior-task accuracy for (a) ℓ_1 and (b) ℓ_2 penalty function.* Target task is Countdown, dashed line shows target- and prior task accuracies for ES without AWD, dotted line shows prior-task accuracy of the base model. For both penalty functions, there is an intermediate range of λ that preserves target-task performance while mitigating forgetting.

6 Analyzing Model Changes under ES with Random Walk Mitigation

According to Eq. (2), increasing the population size reduces the expected norm of the cumulative weight deviation, induced by random walk in the directions weakly constrained by the target task. AWD was introduced with the same objective of reducing such random walk. Therefore, the effect of AWD as well as large population sizes on the weight updates is investigated in this section. Complementarily, an analysis of KL-divergence across tasks is provided, assessing how differences in the magnitude of weight changes translate into changes in the output distribution.

Weight updates. Fig. 6 shows the euclidean norm of changes in the weights $||\boldsymbol{\theta}_t - \boldsymbol{\theta}_0||_2$ for each iteration t throughout training. Consistent with Abdi et al. [2026], ES with a population size of 30 produces updates (cumulative weight changes) with norms roughly two orders of magnitude larger than those for GRPO. Increasing the population size to 128 reduces the magnitude of the update norm by about half; however, even at a population size of 256, the update norm remains more than an order of magnitude higher.

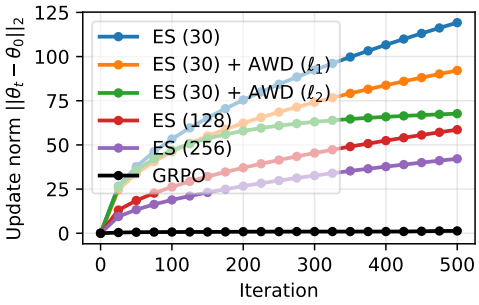


Figure 6: *Norm* $||\boldsymbol{\theta}_t - \boldsymbol{\theta}_0||_2$ *of weight updates.* Larger ES populations and AWD reduce update norms. ES + AWD (ℓ_2) converges to levels similar to ES with population size 128. GRPO updates remain over an order of magnitude smaller.

In contrast, applying AWD to ES with a population size of 30 substantially reduces the update norm, demonstrating its ability to limit random walk in weight space. For ℓ_2 updates, the norm initially matches the case without AWD, but eventually converges to a level comparable to that observed with a population size of 128. Nevertheless, a substantial gap to GRPO remains, raising the question of whether the high-norm updates induced by ES lead to meaningful shifts in LLM behavior.

Distributional shift. Fig. 7 compares the KL-divergence $\mathrm{KL}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}_T})$ between the base model and the final model after training on the Countdown task. Consistent with the trends observed in Fig. 6, ES with a small population size (30) induces substantially larger distributional shifts on prior tasks than GRPO, most prominently on HellaSwag and PIQA. Increasing the population size systematically reduces this divergence, indicating that larger populations mitigate the extent to which ES deviates from the base model, roughly matching the level of KL-divergence exhibited on prior tasks by GRPO. A similar effect is achieved by introducing AWD, especially under ℓ_2 penalty function. Thus, even though the update norm is still much higher than for GRPO, their induced distributional shift as measured by the KL-divergence is comparable.

A markedly different pattern emerges on the target task. GRPO exhibits much higher KL-divergence on Countdown, whereas all ES variants maintain comparatively low divergence despite achieving the same accuracy. While Shenfeld et al. [2025] reported that RL methods such as GRPO adapt to target tasks with much lower KL divergence than supervised fine-tuning, the results show that ES reduces divergence on the target task even further. This observation matches the studies in Qiu et al. [2025].

Overall, the results show that while higher population sizes and AWD substantially reduce the magnitude of the weight change, there still remains a large gap between GRPO and ES. However, GRPO and ES with AWD or high population size lead to similar distributional shift on prior tasks. Furthermore, GRPO induces much greater distributional shift on the target task, albeit having similar accuracy. The conclusion is that it is the randomness of the drift unconstrained by the target task that leads to prior task forgetting for ES, rather than the magnitude of updates alone.

7 Discussion

This paper analyzed prior-task forgetting for ES fine-tuning and introduced AWD to mitigate it. The results show that the extent of such forgetting is mostly driven by random walk behavior. Furthermore, it was found that forgetting is not limited to ES, but can also arise for GRPO. For settings where the issue arises, increasing the population size is an effective mitigation, albeit at increased computational expense. The proposed AWD method is equally effective even with small population sizes, and thus more efficient to employ in practice. Finally, both AWD and higher population sizes decrease the

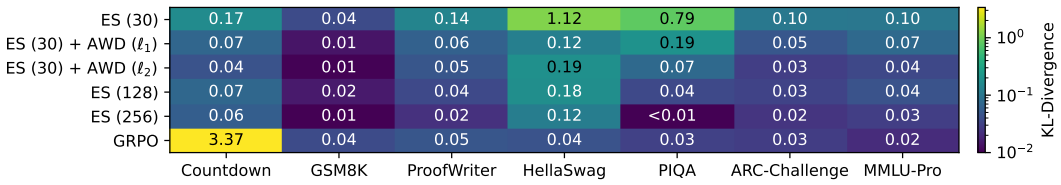


Figure 7: *KL-divergence* $\mathrm{KL}(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}_T})$ *between base and final model trained on Countdown.* ES with a small population size (30) causes larger shifts on prior tasks than GRPO; increasing population size or adding AWD mostly closes this gap. On the target task Countdown, GRPO shows very high divergence while it remains low for all ES variants, despite all having similar target task accuracy.

norm of weight updates. While they are still an order of magnitude higher than for GRPO, they induce similar distributional shift on the prior tasks, suggesting that the norm of weight updates alone does not determine forgetting.

Limitations. AWD incurs a small additional compute cost for the penalty term. However, this overhead is less significant with larger population sizes, because it is applied only once per iteration. Unlike standard ES, AWD requires access to the original weights each iteration, increasing memory usage when stored on the GPU or runtime when streamed from RAM. The experiments were performed using the second option, where using AWD increased the runtime about 1-2%.

Moreover, this study focused on prior-task forgetting in verifiable domains. Extending this analysis to alignment and safety degradation—and testing whether AWD preserves these properties, not just prior task accuracy—is an important direction for future work.

The blessing and curse of random walks. The effectiveness of AWD and larger population sizes point to random walk in directions of weight space weakly coupled to the target task as the primary driver of forgetting. This random walk reflects a fundamental difference between ES and RL: as noted by Hoy et al. [2026], when the reward signal diminishes, RL remains stationary in weight space while ES continues to explore. While AWD restricts this behavior to preserve prior-task performance, the same random walk may allow ES to escape local optima that RL stays trapped in, which is an potential benefit worth investigating empirically in future work.

The choice of penalty function. The results of the comparison between ℓ_1 and ℓ_2 penalty functions in Sec. 5 showed that ℓ_2 is slightly more robust in practice and led to a smoother decline of target task performance when it was set too high. However, induced sparsity of weight updates under ℓ_1 could potentially be used to improve target task performance. Setting λ smaller than the critical magnitude to retain prior-task performance systematically improved target task performance.

While widely used, ℓ_1 and ℓ_2 are only two instances of penalty functions. For instance, combining the two as the Huber penalty function [Huber, 1964] could combine the benefits of both. Future work should investigate the effect of different types of penalty functions on prior-task forgetting under ES.

A scalable path to lifelong learning. Given the results on mitigating prior-task forgetting, ES is well suited for developing agentic systems that continually learn during deployment. Because ES relies only on forward passes, it can fully exploit throughput-optimized hardware without a separate training stack, making it a compelling method for scalable, continually improving systems.

8 Conclusion

Previous works pointed out that ES fine-tuning, while competitive for target task performance, suffers from prior task forgetting [Abdi et al., 2026, Hoy et al., 2026]. This work shows that prior task forgetting during ES fine-tuning is largely avoidable, either intrinsically through larger population sizes or through adding AWD to the base algorithm. This positions ES well for continual learning systems deployed in the real world, where the simplicity and scalability of ES pose unique advantages.

Acknowledgments

We thank Akshat Gupta and Immanuel Abdi for helpful discussions on reproducing their results.

References

Immanuel Abdi, Akshat Gupta, Micah Mok, Alexander Lu, Nicholas Lee, and Gopala Anumanchipalli. Evolutionary strategies lead to catastrophic forgetting in LLMs. *arXiv*, 2601.20861, 2026.

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

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting. *arXiv*, 2510.18874, 2025.

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

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv*, 2110.14168, 2021.

Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

DeepSeek-AI. Deepseek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, page 633–638, 2025.

Yulu Gan and Phillip Isola. Neural thickets: Diverse task experts are dense around pretrained weights. *arXiv*, 2603.12228, 2026.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (SoS): Learning to search in language. *arXiv*, 2404.03683, 2024.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 2001.

Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, 1988.

William Hoy, Binxu Wang, and Xu Pan. Matching accuracy, different geometry: Evolution strategies vs. GRPO in LLM post-training. *arXiv*, 2604.01499, 2026.

Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 1964.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Daria Korotyshova, Boris Shaposhnikov, Alexey Malakhov, Alexey Khokhulin, Nikita Surnachev, Kirill Ovcharenko, George Bredis, Alexey Gorbatovski, Viacheslav Sinii, and Daniil Gavrilov. ESSA: Evolutionary strategies for scalable alignment. *arXiv*, 2507.04453, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Song Lai, Haohan Zhao, Rong Feng, Changyi Ma, Wenzhuo Liu, Hongbo Zhao, Xi Lin, Dong Yi, Qingfu Zhang, Hongbin Liu, Gaofeng Meng, and Fei Zhu. Reinforcement fine-tuning naturally mitigates forgetting in continual post-training. *arXiv*, 2507.05386, 2026.

Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2018.

Qiyao Liang, Jinyeop Song, Yizhou Liu, Jeff Gore, Ila Fiete, Risto Miikkulainen, and Xin Qiu. The blessing of dimensionality in LLM fine-tuning: A variance-curvature perspective. *arXiv*, 2602.00170, 2026.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, 2017.

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

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*. Academic Press, 1989.

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. TinyZero. https://github.com/Jiayi-Pan/TinyZero, 2025. Accessed: 2025-01-24.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

Xin Qiu, Yulu Gan, Conor F. Hayes, Qiyao Liang, Yinggan Xu, Roberto Dailey, Elliot Meyerson, Babak Hodjat, and Risto Miikkulainen. Evolution strategies at scale: LLM fine-tuning beyond reinforcement learning. *arXiv*, 2509.24372, 2025.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv*, 2412.15115, 2025.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Problemata (Stuttgart). Frommann-Holzboog, 1973.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv*, 1606.04671, 2016.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv*, 1703.03864, 2017.

Bidipta Sarkar, Mattie Fellows, Juan Agustin Duque, Alistair Letcher, Antonio León Villares, Anya Sims, Clarisse Wibault, Dmitry Samsonov, Dylan Cope, Jarek Liesen, Kang Li, Lukas Seier, Theo Wolf, Uljad Berdica, Valentin Mohl, Alexander David Goldie, Aaron Courville, Karin Sevegnani, Shimon Whiteson, and Jakob Nicolaus Foerster. Evolution strategies at the hyperscale. *arXiv*, 2511.16652, 2025.

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

Hans-Paul Schwefel. *Numerische Optimierung von Computermodellen mittels der Evolutionsstrategie*. Interdisciplinary Systems Research. Springer Basel AG, 1977.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. Fine-tuned language models are continual learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2022.

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

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. RL's Razor: Why online reinforcement learning forgets less. *arXiv*, 2509.04259, 2025.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv*, 2409.19256, 2024.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv*, 1909.08053, 2019.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 2021.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2008.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 2014.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ying Shan, and Ping Luo. LLaMA pro: Progressive LLaMA with block expansion. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024.

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

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2017.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. PyTorch FSDP: Experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, page 3848–3860, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

A Broader Impact

This work contributes to improving the reliability of LLM fine-tuning by addressing prior-task degradation during post-training. It is shown that higher population sizes, as well as the introduced method AWD are effective in mitigating this issue. This has positive implications for real-world deployment, where models must adapt to new tasks without losing existing capabilities. Reducing unintended degradation also lowers the need for repeated retraining, potentially decreasing computational costs and environmental impact.

However, enabling more effective continual learning may also accelerate the deployment of increasingly autonomous systems that adapt over time. This raises concerns around monitoring, evaluation, and control, as models could evolve in unintended ways if not properly constrained. While AWD has been found effective to mitigate performance drift relative to the initial model, its ability to preserve alignment or safety properties during fine-tuning has not been tested yet, which is an important direction for future work.

Finally, the findings of this paper highlight that fine-tuning dynamics play a central role in distribution shift. This underscores the importance of developing robust evaluation protocols and safeguards under continuous model updates.

B Background on Group Relative Policy Optimization

Group Relative Policy Optimization (GRPO) [Shao et al., 2024] is an RL–based method for fine-tuning LLMs. It is a Proximal Policy Optimization (PPO) [Schulman et al., 2017] variant that avoids the need for an explicit value function. Given an input sequence x, GRPO samples a group of response sequences $\{y_k\}_{k=1}^{K}$ from the current policy (LLM) $p_{\boldsymbol{\theta}}$ and computes rewards $R(y_k)$. Instead of estimating advantages via a learned critic, GRPO normalizes rewards within the sampled group to obtain advantages $\hat{A}_k = \frac{R(y_k)-\mu_K}{\sigma_K}$, where μ_K and σ_K are the mean and standard deviation over the sampled group of responses. The policy is then updated using a clipped surrogate objective as in PPO [Schulman et al., 2017]:

$$J_{\text{GRPO}}(\boldsymbol{\theta}) = \mathbb{E}\left[\frac{1}{K}\sum_{k=1}^{K}\min\left(\rho_k(\boldsymbol{\theta})\hat{A}_k,\ \text{clip}(\rho_k(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_k\right)\right], \tag{5}$$

where $\rho_k(\boldsymbol{\theta}) = \frac{p_{\boldsymbol{\theta}}(y_k|x)}{p_{\boldsymbol{\theta}_{\text{old}}}(y_k|x)}$ is the likelihood ratio of the output sequence between the current model and a previous version of the model. Usually, Eq. (5) is extended by a KL-divergence penalty term between the current policy and the original policy, acting as regularization. The weights $\boldsymbol{\theta}$ of the policy are then updated via backpropagation to maximize the objective. GRPO preserves the effectiveness of PPO while being simpler and more efficient due to the removal of the value model. Therefore, it has emerged as a strong default for LLM post-training, particularly in reasoning tasks. However, implementations of RL-based pipelines are complex, requiring separate systems for sampling output trajectories [Kwon et al., 2023, Zheng et al., 2024] and for training the model via backpropagation [Zhao et al., 2023, Shoeybi et al., 2019], increasing engineering overhead and limiting scalability. Scaling these systems to multiple processing units or even compute nodes is a major engineering challenge, as gradients and optimizer states need to be synchronized between different processing units. In contrast, ES requires just a single system for sampling output trajectories and scales without much engineering effort as solely scalar information about seeds and rewards needs to be exchanged between processing units.

C Connection to Regularization-Based Mitigation Methods for Forgetting

The works of Kirkpatrick et al. [2017] and Zenke et al. [2017] have demonstrated that regularizing changes to the weights when training on a new task is effective to mitigate prior task forgetting of neural networks. In the following, their connection to AWD is highlighted, and it is discussed how AWD can be interpreted through the lens of their papers motivations.

Connection to Kirkpatrick et al. [2017]. Elastic Weight Consolidation (EWC) [Kirkpatrick et al., 2017] considers a quadratic penalty towards the original weights $\boldsymbol{\theta}_0$, that is weighted by their

importance on prior tasks. EWC is motivated through the Bayesian framework: The relevance of $\boldsymbol{\theta}$ w.r.t the training data $\mathcal{D}$ is given by its posterior probability $p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}$ (Bayes rule). Importantly, the log probability of the data given the weights $\log p(\mathcal{D} \mid \boldsymbol{\theta})$, their log-likelihood, is nothing else than the negative of the standard cross entropy loss function $-\mathcal{L}(\boldsymbol{\theta})$. Assuming the data can be split into two independent parts that define tasks A and B and applying the logarithm to Bayes rule, allows to rearrange to

$$\log p(\boldsymbol{\theta} \mid \mathcal{D}) = \log p(\mathcal{D}_B \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta} \mid \mathcal{D}_A) - \log p(\mathcal{D}_B) \,, \tag{6}$$

Thus all information about task A is provided by the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D}_A)$. It thus provides information, which weights were important for task A, yet the posterior is generally intractable to compute exactly. EWC follows the ideas of the Laplace posterior approximation in assuming a Gaussian centered around the original weights $\boldsymbol{\theta}_0$, i.e. weights that have been optimized on task A. The precision of the Gaussian considers a diagonal approximation for tractability (the covariances between weights are neglected), which is given by the diagonal of the Fisher information matrix F for the task A, thus $F_i = -\mathbb{E}\left[\frac{\partial^2}{\partial \theta_i^2} \log p(\mathcal{D}_A \mid \boldsymbol{\theta}) \mid \boldsymbol{\theta}\right]$. Thus, they implement the loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_B(\boldsymbol{\theta}) + \frac{\lambda}{2} \sum_{i=1}^{d} F_i(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{0,i})^2 \,. \tag{7}$$

This is related to the definition of the regularized objective function in Eq. (3), where EWC considers the ℓ_2 penalty function. While for gradient based methods, it is applicable to compute the regularization term as part of the loss function and automatically change the update of weights through backpropagation, for ES this is not applicable and it is necessary to directly implement AWD in the weight update equation. Furthermore, in the setting considered in this paper, there is no fixed prior task A, but countless possible prior tasks that the LLM could be used for. Considering the Bayesian formulation that motivated EWC, AWD models the posterior $p(\boldsymbol{\theta} \mid \mathcal{D}_A)$ with a uniform distribution, following the maximum entropy principle, arriving at a uniform weighting. If there is a certain set of tasks that one cares about, AWD could be further improved by modeling F as done by EWC.

Connection to Zenke et al. [2017]. The regularization approach of Zenke et al. [2017] arrives at a similar loss term as EWC, yet with different motivation and interpretation of the weighting term Ω_i, and is given by

$$\mathcal{L}(\theta) = \mathcal{L}_B(\boldsymbol{\theta}) + \frac{\lambda}{2} \sum_{i=1}^{d} \Omega_i(\boldsymbol{\theta}_i - \boldsymbol{\theta}_{0,i})^2 \,, \tag{8}$$

where the formulation compared to the original paper is adapted to highlight the similarity to EWC and AWD. Zenke et al. [2017] consider a sequence of tasks, which is presented here in the special case of a two task setting A followed by B. Their weighting term is given by $\Omega_i = \frac{\omega_i^A}{\Delta \boldsymbol{\theta}_i^A}$, where ω_i^A is the parameter-specific contribution to the change in loss for task A and $\Delta \boldsymbol{\theta}_i^A$ denotes how much the weights changed for learning task A. It thus takes the whole optimization trajectory on task A into account for determining the weighting, not only the local sensitivity around the final model weights as in EWC. In practice, ω_i^A is approximated as the running sum of the product between the current gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i}$ and the update to the parameter $\frac{\partial \boldsymbol{\theta}_i}{\partial t}$.

The connection to AWD is similar as between EWC and AWD, yet the interpretation of using a uniform weighting for Ω_i differs. Intuitively, having uniform Ω_i means that every weight contributed equivalently to minimizing the loss on all prior tasks. Again, if there are some tasks that one cares about to not loose performance, a non-uniform weighting for AWD can be introduced as in Zenke et al. [2017] by estimating Ω_i on those tasks.

D Detailed Description of Experimental Setup

Detailed information on the hyperparameters of ES and GRPO, on the models used for evaluation and descriptions of the considered tasks, including examples, answer format, and extraction logic is provided in the following.

D.1 Hyperparameters for ES and GRPO

Tuning strategy. For both ES and GRPO, an initial sweep of the most important hyperparameters on the Countdown task for the Qwen2.5 3B Instruct model was conducted. The parameters selected in Qiu et al. [2025] and Abdi et al. [2026] proved to be very effective. Then, these parameters were tested on the other target tasks with the same model, which led to good results, with comparable performance gains for both ES and GRPO. A small additional sweep was performed to confirm that these parameters were competitive, which proved to be the case. While the target task performance is not the main focus of our study, as this paper mostly studies prior-task performance, it was nevertheless made sure that good models are obtained for a fair comparison.

ES. The main hyperparameters selected for ES are as follows: $\sigma = 0.001$, $\alpha = \sigma/2$, population size = 30 (except where otherwise specified by experiments), 500 training iterations.

GRPO. The main hyperparameters selected for GRPO are as follows: lr = 1e-6, batch size = 32, KL-loss coefficient = 0.001 (applied to loss, not reward), entropy-loss coefficient = 0, number of rollouts = 30, 500 total epochs.

D.2 Models

Qwen-2.5 Instruct is distributed under the Qwen Research Licence Agreement. [1] The official weights, tokenizer, and chat template as provided through HuggingFace `transformers` are used.

Llama-3.2 Instruct is distributed under the Llama 3.2 Community License Agreement. [2] The official weights, tokenizer, and chat template as provided through HuggingFace `transformers` are used.

D.3 Tasks

Countdown considers arithmetic reasoning, where the model must combine a set of given numbers using basic operations to reach a target value.

Published under Apache License 2.0 in the official codebase of Gandhi et al. [2024].

Countdown Example

Using the numbers [44, 19, 35], create an equation that equals 98.

Answer format	Extraction logic
`<answer>...</answer>`	For correctness, all `<answer>...</answer>` blocks are extracted and only the last one is used; if none exists, reward is zero. Within that block, the text must be non-empty and consist only of `[0-9+*/() ]`, otherwise the reward is zero. Then all integer substrings are extracted, and their multiset must exactly match the provided numbers, i.e. every number must appear exactly once. Finally, the expression is evaluated, and the reward is one only if the result matches the target within a tolerance of `1e-5`; any evaluation error gives zero reward.

GSM8K focuses on grade-school math word problems that require multi-step numerical reasoning and understanding of natural language.

Published under MIT license in the official codebase of [Cobbe et al., 2021].

[1] https://huggingface.co/Qwen/Qwen2.5-3B-Instruct/blob/main/LICENSE
[2] https://huggingface.co/meta-llama/Llama-3.2-1B/blob/main/LICENSE.txt

Example

Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?

Answer format	Extraction logic
`#### <number>`	Only the last 4096 characters of the response are searched; if `</think>` appears there, only the text after the last such tag is used. Extraction then tries, in order: last `#### ...` match, last `\boxed{...}` match, last `<answer>...</answer>` match, otherwise the full search region. The extracted text is parsed numerically by stripping commas, `$`, and `%`; if it is an exact fraction `a/b`, that fraction is evaluated, otherwise the first numeric substring is used. Reward is zero if no number can be parsed or if it does not match the target within a tolerance of `1e-6`.

ProofWriter evaluates logical reasoning, requiring the model to derive conclusions from a set of facts and rules using formal inference.

There was no license information provided in the paper, nor for the HuggingFace dataset source. [3]

Example

Theory: The dog is not blue. If someone is green and not cold, then they are round.
Question: The dog is not blue.

Answer format	Extraction logic
`Answer: <True/False/Unknown>`	Extraction tries, in order: the last multiline match of `Answer: <word>` anywhere in the response; otherwise the last non-empty line, matched leniently as optional `Final answer` or `Answer` followed by a word with optional trailing punctuation; otherwise the last occurrence anywhere of `true`, `false`, or `unknown`; otherwise the whole response is canonicalized. Canonicalization trims whitespace, strips trailing punctuation `.!?,:;`, lowercases, and maps only `true/false/unknown` to the valid labels. The reward is zero if canonicalization fails or the label does not exactly match the target.

HellaSwag focuses on commonsense reasoning, where the model selects the most plausible continuation of a given situation.

Published under MIT license in the official codebase of Zellers et al. [2019].

Example

Activity: Roof shingle removal
Context: A man is sitting on a roof. he
Endings:
0. is using wrap to wrap a pair of skis.
1. is ripping level tiles off.
2. is holding a Rubik's Cube.
3. starts pulling up roofing on a roof.

PIQA evaluates physical commonsense understanding, requiring the model to choose solutions that are feasible in real-world scenarios.

Published under Academic Free License v.3.0 in the official codebase of Bisk et al. [2020].

[3] https://huggingface.co/datasets/tasksource/proofwriter

Answer format	Extraction logic
<0/1/2/3>	The strict format is satisfied only if the entire output is exactly one digit in 0-3 up to surrounding whitespace. For answer extraction, however, the first occurrence of any character in [0-3] anywhere in the output is used. There are no further fallbacks. The reward is zero if no such digit is found or if it does not equal the gold label.

Example

Goal: How do I ready a guinea pig cage for its new occupants?
1. Provide the guinea pig with a cage full of a few inches of bedding made of ripped paper strips, you will also need to supply it with a water bottle and a food dish.
2. Provide the guinea pig with a cage full of a few inches of bedding made of ripped jeans material, you will also need to supply it with a water bottle and a food dish.

Answer format	Extraction logic
<1/2>	The strict format is satisfied only if the entire output is exactly 1 or 2 up to surrounding whitespace. For answer extraction, the first standalone 1 or 2 found via word boundaries is used without further fallbacks. The reward is zero if no such label is found or if it does not equal the gold label.

ARC-Challenge consists of difficult science questions that test reasoning and knowledge beyond simple retrieval.

Listed as CC BY-SA 4.0 according to HuggingFace dataset source. [4]

Example

An astronomer observes that a planet rotates faster after a meteorite impact. Which is the most likely effect of this increase in rotation?
A. Planetary density will decrease.
B. Planetary years will become longer.
C. Planetary days will become shorter.
D. Planetary gravity will become stronger.

Answer format	Extraction logic
The answer is (<A/B/C/D/E>)	The response is first uppercased. Extraction then tries, in order: the first match of ANSWER IS (<A/B/C/D/E>), otherwise the first match of ANSWER IS <A/B/C/D/E>, otherwise the first standalone letter in A-E anywhere in the response. The reward is zero if no valid choice letter is found or if it does not equal the gold answer.

MMLU-Pro measures broad academic knowledge and reasoning across a wide range of professional and academic subjects.

Published under Apache License 2.0 in the official codebase of Wang et al. [2024].

[4] https://huggingface.co/datasets/allenai/ai2_arc

Example

Typical advertising regulatory bodies suggest, for example, that adverts must not: encourage _____, cause unnecessary _____ or _____, and must not cause _____ offence.
A. Safe practices, Fear, Jealousy, Trivial
B. Unsafe practices, Distress, Joy, Trivial
C. Safe practices, Wants, Jealousy, Trivial
D. Safe practices, Distress, Fear, Trivial
E. Unsafe practices, Wants, Jealousy, Serious
F. Safe practices, Distress, Jealousy, Serious
G. Safe practices, Wants, Fear, Serious
H. Unsafe practices, Wants, Fear, Trivial
I. Unsafe practices, Distress, Fear, Serious

Answer format	Extraction logic
`The answer is (<A/.../J>)`	The response is first uppercased. Extraction then tries, in order: the first match of `ANSWER IS (<A/.../J>)`, otherwise the first match of `ANSWER IS <A/.../J>`, otherwise the first standalone letter in `A-J` anywhere in the response. The reward is zero if no valid choice letter is found or if it does not equal the gold answer.

E Detailed Results on Individual Prior-Task Accuracies

Accuracies for the base models on each task are provided in Tab. 3.

Furthermore, per-task accuracies for ES and GRPO optimization for GSM8K and ProofWriter as target tasks are shown in Fig. 8 and Fig. 9. Additionally, the individual performances per dataset of ES and ES+AWD, which was shown aggregated over datasets in Fig. 1, is shown in Fig. 10.

F Additional Results on Applying AWD

This section provides additional results for applying AWD on different target datasets and model types as follows. Results for training on ProofWriter as target task with Qwen-2.5 3B Instruct as model are provided in Tab. 4. Results for training on Countdown as target task with Llama-3.2 3B Instruct as model are provided in Tab. 5. Results for training on ProofWriter as target task with Llama-3.2 3B Instruct as model are provided in Tab. 6.

Table 3: Individual task accuracies of original models before applying ES or GRPO. ES and GRPO implementations use different precisions in their original implementations, which was kept as is to stay consistent with prior work. ES: `float16`, GRPO: `bfloat16`. This leads to slight performance differences even for the base model under greedy decoding. Top: `float16` (ES). Bottom: `bfloat16` (GRPO).

Model	Countdown	GSM8K	ProofWriter	HellaSwag	PIQA	ARC-C	MMLU-Pro
Qwen-2.5 1.5B Instruct	5.3	58.2	53.6	57.5	71.7	75.3	31.5
Qwen-2.5 3B Instruct	11.5	83.8	44.2	63.7	81.2	85.2	41.6
Qwen-2.5 7B Instruct	25.5	91.2	67.6	83.1	86.0	91.1	57.3
Llama-3.2 3B Instruct	4.0	75.0	38.4	41.8	66.4	77.5	20.4
Qwen-2.5 1.5B Instruct	4.9	58.2	51.9	56.8	71.4	75.4	32.1
Qwen-2.5 3B Instruct	10.0	84.0	45.1	64.0	81.3	84.8	41.9
Qwen-2.5 7B Instruct	24.5	90.2	67.9	83.2	86.1	91.4	56.9
Llama-3.2 3B Instruct	2.2	75.2	41.7	40.6	68.2	78.6	21.7

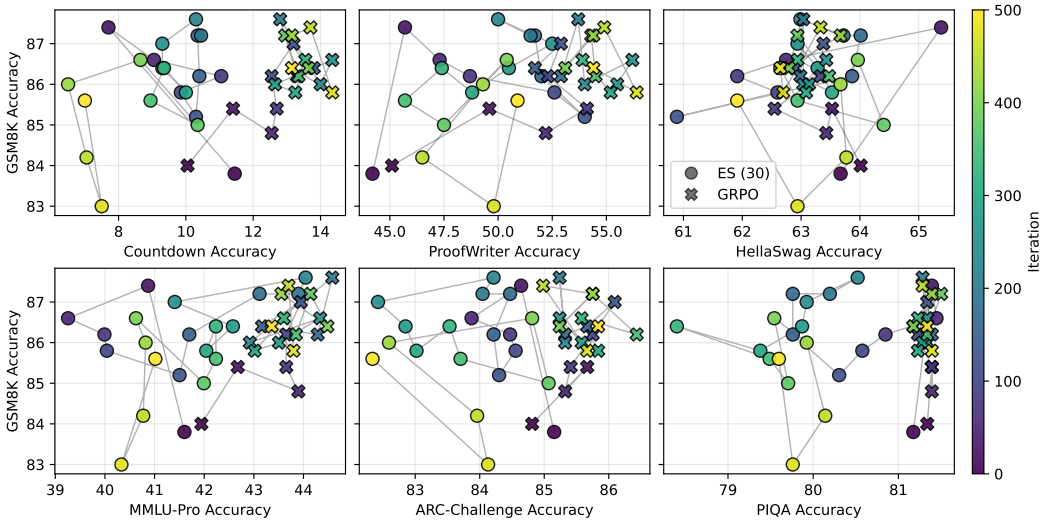


Figure 8: *Individual prior task accuracies training on GSM8K as target task.* There is no systematic forgetting observed for ES nor for GRPO.

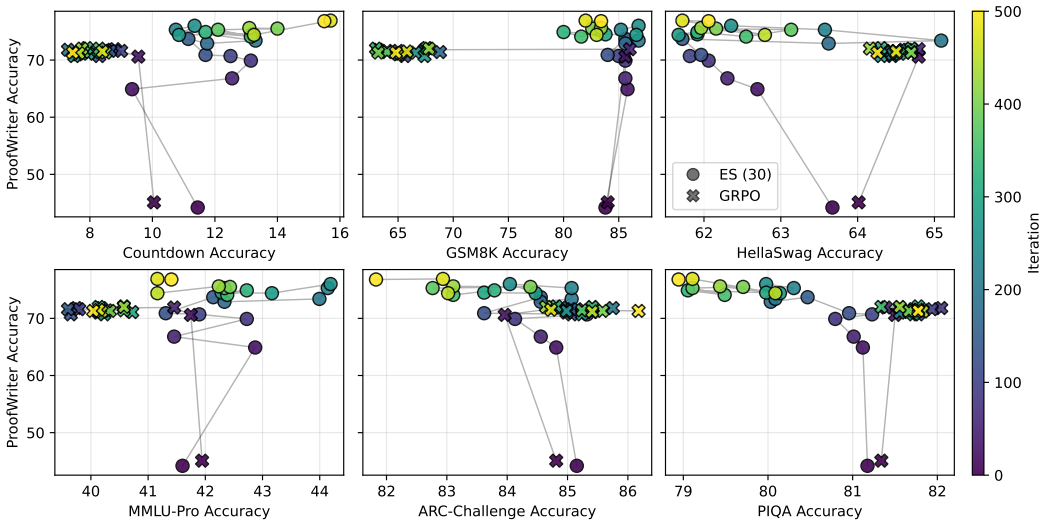


Figure 9: *Individual prior task accuracies training on ProofWriter as target task.* Strong forgetting is observed for GRPO on GSM8K as a prior task, as well as moderate forgetting on Countdown and MMLU-Pro. For ES, mild forgetting is observed for HellaSwag, ARC-Challenge, and PIQA.

The penalty factor λ was not tuned for individual configurations, showing that the values suggested by the ablation in the main paper—for ℓ_1: $\lambda = 0.01$ and for ℓ_2: $\lambda = 10.0$—transfer well between different target task and model type configurations.

G KL-Divergence for Different Training Tasks

In Sec. 4 in the main paper, the KL-divergence on the target task Countdown and the remaining prior tasks was evaluated for ES under different population sizes, as well as when combined with AWD. Complementary to those results, Fig. 11 shows the KL-divergence for GRPO and ES with population size 30 for the three different target tasks. In line with Qiu et al. [2025], the results show lower KL-divergence for ES when evaluated on the target task. For the prior-tasks not trained on, GRPO generally attains lower KL-divergence than ES with low population size. This might be explained

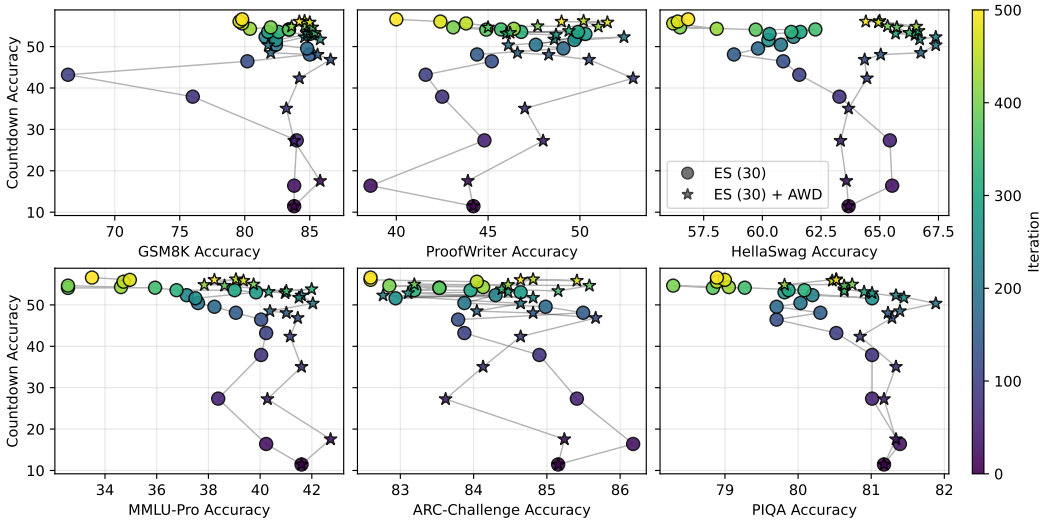


Figure 10: *Individual prior task accuracies training on Countdown as target task with and without AWD.* While forgetting is observed for standard ES, there is essentially no forgetting in any task with ES + AWD (ℓ_2), except a slight degradation on MMLU-Pro.

Table 4: *Change in target (ProofWriter) and prior task accuracies using AWD with ℓ_1/ℓ_2 decay on Qwen-2.5 3B Instruct.* The average is calculated over accuracies in all prior tasks, thus excluding ProofWriter. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. For ProofWriter, the baseline without AWD already has relatively low forgetting. Applying AWD with ℓ_1 ($\lambda = 0.01$) leads to similar results, improving for some prior tasks, but leading to a bit more performance degradation for others. Applying AWD with ℓ_2 ($\lambda = 10.0$) leads to improvements for all prior tasks except Countdown (which has a high std).

Run	Countdown	GSM8K	**ProofWriter**	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
ES (30)	$-0.6_{(4.8)}$	$-0.3_{(0.3)}$	$+35.6_{(2.7)}$	$-1.4_{(0.2)}$	$-1.9_{(0.8)}$	$-2.5_{(1.6)}$	$-0.7_{(0.5)}$	$-1.3_{(0.8)}$
ES+AWD (L1)	$+0.4_{(3.0)}$	$+0.7_{(0.8)}$	$+34.6_{(0.8)}$	$-3.6_{(2.2)}$	$-1.1_{(0.5)}$	$-1.0_{(0.3)}$	$-2.7_{(3.5)}$	$-1.2_{(0.4)}$
ES+AWD (L2)	$-2.3_{(3.4)}$	$+1.3_{(0.6)}$	$+34.4_{(0.4)}$	$-1.1_{(3.1)}$	$-0.9_{(1.1)}$	$-0.5_{(0.2)}$	$+0.5_{(0.8)}$	$-0.5_{(0.6)}$

Table 5: *Change in target (Countdown) and prior task accuracies using AWD with ℓ_1/ℓ_2 decay on Llama-3.2 3B Instruct* . The average is calculated over accuracies in all prior tasks, thus excluding Countdown. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. Applying AWD with ℓ_1 ($\lambda = 0.01$) leads to improvements across all prior tasks except ProofWriter, which has relatively high variance. Applying AWD with ℓ_2 ($\lambda = 10.0$) leads to even stronger improvements generally, again except for ProofWriter where it decreases prior task performance.

Run	**Countdown**	GSM8K	ProofWriter	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
ES (30)	$+46.9_{(1.0)}$	$-5.1_{(3.4)}$	$+3.9_{(6.0)}$	$-12.6_{(2.0)}$	$-3.3_{(3.3)}$	$-4.2_{(1.1)}$	$-5.4_{(3.8)}$	$-4.4_{(1.7)}$
ES+AWD (L1)	$+45.0_{(0.4)}$	$+1.9_{(2.1)}$	$-0.1_{(5.1)}$	$-4.0_{(0.9)}$	$-1.6_{(2.2)}$	$-1.7_{(0.3)}$	$-5.3_{(2.5)}$	$-1.8_{(1.0)}$
ES+AWD (L2)	$+44.6_{(0.1)}$	$+2.9_{(2.1)}$	$-1.7_{(3.6)}$	$-2.8_{(0.7)}$	$+1.6_{(2.2)}$	$-0.9_{(0.6)}$	$-3.5_{(3.5)}$	$-0.7_{(1.0)}$

by the more targeted, low-magnitude changes induced by GRPO. For the target task, those lead to massive changes in the output distribution, while they less impact the output distribution of dissimilar tasks, while ES changes the output distribution less for the particular tasks, but induces more global change. Notably though, the results in the main paper showed that the KL-divergence of ES on prior-tasks reduces with higher population size and when combined with AWD.

Table 6: *Change in target (ProofWriter) and prior task accuracies using AWD with ℓ_1/ℓ_2 decay for Llama-3.2 3B Instruct.* The average is calculated over accuracies in all prior tasks, thus excluding ProofWriter. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. Applying AWD with ℓ_1 ($\lambda = 0.01$) mostly decreases prior task forgetting, especially on HellaSwag, ARC and MMLU-Pro where it was most severe without AWD. Applying AWD with ℓ_2 ($\lambda = 10.0$) leads to similar improvements, yet also a substantial drop for PIQA - with high variance.

Run	Countdown	GSM8K	**ProofWriter**	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
ES (30)	$+0.8_{(1.7)}$	$-4.7_{(0.8)}$	$+30.5_{(1.6)}$	$-5.4_{(4.5)}$	$-3.0_{(2.5)}$	$-6.7_{(0.6)}$	$-4.4_{(7.2)}$	$-3.9_{(2.4)}$
ES+AWD (L1)	$+0.4_{(0.8)}$	$-1.1_{(2.7)}$	$+30.8_{(1.6)}$	$-0.1_{(2.4)}$	$-3.4_{(4.8)}$	$-1.9_{(0.8)}$	$-3.7_{(3.3)}$	$-1.6_{(1.7)}$
ES+AWD (L2)	$+0.9_{(0.7)}$	$-0.6_{(1.3)}$	$+29.3_{(1.4)}$	$-2.1_{(4.0)}$	$-4.5_{(4.5)}$	$-1.0_{(1.0)}$	$+2.9_{(4.2)}$	$-0.8_{(1.7)}$

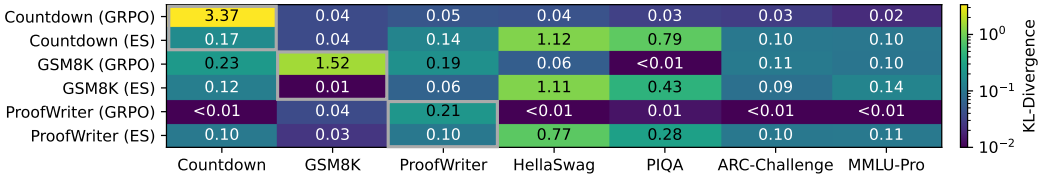


Figure 11: *KL-divergence $KL(p_{\boldsymbol{\theta}_0}||p_{\boldsymbol{\theta}_T})$ for different target tasks across prior tasks for ES (population size 30) and GRPO.* The y-axis shows training task and fine-tuning method, the x-axis shows the task used for evaluation. When evaluating on the target task (grey boxes), ES (30) attains lower KL-divergence than GRPO, though both have similar accuracies. For the prior-tasks, GRPO mostly has lower KL-Divergences then ES (30).

H Detailed Results on Dependence on Target Task and Model Type

Detailed results comparing ES with a population size of 30 with GRPO are provided as follows. Tab. 7 shows accuracy change across model families (for the 3B size each), when trained on Countdown as target task. Tab. 8 shows accuracy change across model families (for the 3B size each), when trained on ProofWriter as target task. Tab. 9 shows accuracy change across model size (for Qwen model series), when trained on Countdown as target task. Tab. 10 shows accuracy change across model size (for Qwen model series), when trained on ProofWriter as target task.

I Compute Resources

All experiments were performed on a node with eight H200s. Individual runs for ES on the 3B model on Countdown took around 4 node-hours, the smaller 1.5B model took around three node-hours, and the larger 7B model around six node-hours. For GRPO, a 3B run took around 20 node-hours, a 1.5B run around 14 node-hours, and a 7B run took around 32 node-hours. For ES, runs for GSM8K and ProofWriter require only around 75% of the training time of Countdown, as answer lengths are generally shorter (generating answers for the population is the dominant compute cost for ES). For GRPO, this discrepancy is much smaller as generating the answers is only one part of the overall compute cost, with forward and backward pass for updating the model as well as another forward pass for the KL-regularization being more dominant, leading to more uniform runtimes over different tasks.

Overall, the tracked node-hours for the whole project, including failed runs and initial experiments not included in the final paper, amount to approximately 52 node-days (1248 node-hours). Performing evaluations on the trained models took approximately one extra node-day.

Table 7: *Accuracy change across model families.* The target task is Countdown, and the average is calculated over all other tasks. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. Prior-task forgetting is similar for both the Qwen and Llama 3B model.

Method	**Countdown**	GSM8K	ProofWriter	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
Qwen2.5 (ES)	$+45.3_{(0.3)}$	$-9.5_{(11.5)}$	$-2.0_{(5.7)}$	$-8.2_{(9.0)}$	$-2.8_{(1.0)}$	$-2.0_{(0.5)}$	$-6.5_{(2.6)}$	$-5.2_{(4.7)}$
Qwen2.5 (GRPO)	$+46.9_{(2.4)}$	$+0.5_{(0.9)}$	$+3.5_{(4.3)}$	$+1.1_{(0.2)}$	$+0.2_{(0.2)}$	$+0.1_{(0.4)}$	$-2.3_{(1.7)}$	$+0.5_{(0.9)}$
Llama-3.2 (ES)	$+46.9_{(1.0)}$	$-5.1_{(3.4)}$	$+3.9_{(6.0)}$	$-12.6_{(2.0)}$	$-3.3_{(3.3)}$	$-4.2_{(1.1)}$	$-5.4_{(3.8)}$	$-4.4_{(1.7)}$
Llama-3.2 (GRPO)	$+41.5_{(9.4)}$	$-0.4_{(1.6)}$	$+1.2_{(0.3)}$	$+1.3_{(1.3)}$	$-7.6_{(4.5)}$	$+0.5_{(0.4)}$	$-2.8_{(2.8)}$	$-1.3_{(1.0)}$

Table 8: *Accuracy change across model families.* The target task is ProofWriter, and the average is calculated over all other tasks. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. Prior-task forgetting is similar for ES, but differs for GRPO.

Method	Countdown	GSM8K	**ProofWriter**	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
Qwen2.5 (ES)	$-0.6_{(4.8)}$	$-0.3_{(0.3)}$	$+35.6_{(2.7)}$	$-1.4_{(0.2)}$	$-1.9_{(0.8)}$	$-2.5_{(1.6)}$	$-0.7_{(0.5)}$	$-1.3_{(0.8)}$
Qwen2.5 (GRPO)	$-2.3_{(0.6)}$	$-10.1_{(9.0)}$	$+25.3_{(1.5)}$	$-0.8_{(0.9)}$	$-0.0_{(0.4)}$	$+0.6_{(0.8)}$	$-1.0_{(0.8)}$	$-2.3_{(1.4)}$
Llama-3.2 (ES)	$+0.8_{(1.7)}$	$-4.7_{(0.8)}$	$+30.5_{(1.6)}$	$-5.4_{(4.5)}$	$-3.0_{(2.5)}$	$-6.7_{(0.6)}$	$-4.4_{(7.2)}$	$-3.9_{(2.4)}$
Llama-3.2 (GRPO)	$+3.8_{(2.2)}$	$-0.3_{(2.6)}$	$+24.2_{(1.2)}$	$+6.4_{(2.7)}$	$+0.4_{(1.5)}$	$-0.1_{(0.2)}$	$+1.3_{(6.2)}$	$+1.9_{(0.9)}$

Table 9: *Accuracy change across different model sizes.* The target task is Countdown and the average is calculated over all other tasks. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. Forgetting of ES arises under all model sizes, whereas GRPO does not exhibit pronounced forgetting.

Method	**Countdown**	GSM8K	ProofWriter	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
ES 1.5B	$+36.9_{(9.8)}$	$+4.5_{(5.0)}$	$-3.9_{(2.8)}$	$-9.7_{(3.4)}$	$-3.2_{(1.9)}$	$-4.2_{(5.1)}$	$-2.8_{(1.1)}$	$-3.2_{(0.8)}$
ES 3B	$+45.3_{(0.3)}$	$-9.5_{(11.5)}$	$-2.0_{(5.7)}$	$-8.2_{(9.0)}$	$-2.8_{(1.0)}$	$-2.0_{(0.5)}$	$-6.5_{(2.6)}$	$-5.2_{(4.7)}$
ES 7B	$+35.0_{(2.5)}$	$-0.8_{(1.2)}$	$-8.2_{(8.1)}$	$-3.3_{(1.2)}$	$-2.0_{(0.5)}$	$-1.1_{(0.4)}$	$-4.8_{(1.3)}$	$-3.4_{(1.6)}$
GRPO 1.5B	$+17.3_{(0.1)}$	$+16.8_{(1.0)}$	$+1.0_{(0.5)}$	$-1.1_{(1.2)}$	$+1.9_{(1.0)}$	$-0.3_{(0.1)}$	$+0.4_{(0.4)}$	$+3.1_{(0.2)}$
GRPO 3B	$+46.9_{(2.4)}$	$+0.5_{(0.9)}$	$+3.5_{(4.3)}$	$+1.1_{(0.2)}$	$+0.2_{(0.2)}$	$+0.1_{(0.4)}$	$-2.3_{(1.7)}$	$+0.5_{(0.9)}$
GRPO 7B	$+36.8_{(0.3)}$	$+2.5_{(0.9)}$	$-0.3_{(0.3)}$	$+0.8_{(0.5)}$	$+0.3_{(0.2)}$	$+0.2_{(0.3)}$	$-0.5_{(0.1)}$	$+0.5_{(0.2)}$

Table 10: *Accuracy change across different model sizes.* The target task is ProofWriter and the average is calculated over all other tasks. Statistics are computed over three runs; for baseline accuracies see Tab. 3 in App E. Forgetting of ES arises under all model sizes. GRPO also exhibits forgetting for the two larger model sizes.

Method	Countdown	GSM8K	**ProofWriter**	HellaSwag	PIQA	ARC-C	MMLU-Pro	Average
ES 1.5B	$-0.9_{(2.6)}$	$+0.7_{(16.1)}$	$+9.8_{(0.4)}$	$-7.1_{(2.7)}$	$-0.9_{(2.8)}$	$-2.0_{(0.8)}$	$-2.7_{(1.7)}$	$-2.2_{(2.5)}$
ES 3B	$-0.6_{(4.8)}$	$-0.3_{(0.3)}$	$+35.6_{(2.7)}$	$-1.4_{(0.2)}$	$-1.9_{(0.8)}$	$-2.5_{(1.6)}$	$-0.7_{(0.5)}$	$-1.3_{(0.8)}$
ES 7B	$-2.5_{(3.0)}$	$-1.0_{(0.7)}$	$+10.9_{(1.7)}$	$-3.5_{(1.2)}$	$-2.2_{(0.6)}$	$-0.7_{(0.8)}$	$-5.0_{(0.8)}$	$-2.5_{(0.8)}$
GRPO 1.5B	$-0.9_{(2.7)}$	$+7.8_{(4.2)}$	$+12.4_{(3.5)}$	$-0.2_{(1.3)}$	$+0.9_{(0.4)}$	$-0.1_{(0.3)}$	$+0.3_{(0.9)}$	$+1.3_{(0.3)}$
GRPO 3B	$-2.3_{(0.6)}$	$-10.1_{(9.0)}$	$+25.3_{(1.5)}$	$-0.8_{(0.9)}$	$-0.0_{(0.4)}$	$+0.6_{(0.8)}$	$-1.0_{(0.8)}$	$-2.3_{(1.4)}$
GRPO 7B	$+0.8_{(1.2)}$	$+1.5_{(1.2)}$	$+7.2_{(6.7)}$	$-0.6_{(1.8)}$	$-1.3_{(1.6)}$	$+0.0_{(0.4)}$	$-0.9_{(0.9)}$	$-0.1_{(1.0)}$