What Makes RL Generalize?
A Systematic Study with Gradient Agreement Regularization

[Author Name] stitution¿stitution

[email]

Abstract

Generalization remains a fundamental challenge in deep reinforcement learning (RL). While agents can master specific training environments, they often fail when tested under distribution shift. In this paper, we present a systematic study of RL generalization across three dimensions: algorithm choice (DQN, PPO, ES), training technique (baseline, L2 regularization, Gradient Agreement Regularization), and environment type (dynamics variation in CartPole, layout variation in MiniGrid, and game mechanics variation in MinAtar Space Invaders). We introduce the **Generalization Robustness Score (GRS)**, a metric that captures the full shape of performance degradation under distribution shift, rather than reducing generalization to a single train-test gap. We further propose **Gradient Agreement Regularization (GAR)**, which filters gradient updates by their agreement across training variations to encourage learning of generalizable features. Our experiments across 108 runs reveal that: (1) GAR significantly improves PPO generalization, achieving up to 97% improvement in GRS on CartPole and 103% on MiniGrid, (2) the effectiveness of regularization techniques is algorithm-dependent, with on-policy methods benefiting most, and (3) Evolution Strategies fundamentally cannot solve sparse-reward tasks like MiniGrid navigation. These findings provide actionable insights for practitioners deploying RL agents where environmental variation is inevitable.

1 Introduction

Deep reinforcement learning has achieved remarkable successes in controlled environments, from mastering Atari games to defeating world champions in Go. However, these achievements often mask a critical weakness: trained agents become overly specialized to their training conditions and fail when faced with environmental variations. A robot trained to walk on flat terrain may stumble on slightly uneven ground; a game-playing agent that masters one set of levels may fail on procedurally generated variations.

This brittleness poses a fundamental barrier to deploying RL in real-world applications where environmental variation is the norm. Despite growing awareness of this problem, systematic understanding of what makes RL generalize remains limited. Prior work has demonstrated that overfitting occurs [1], but the community lacks: (1) standardized metrics for measuring generalization quality, (2) techniques specifically designed to improve generalization, and (3) comprehensive comparisons across algorithm families and environment types.

In this paper, we address these gaps through three main contributions:

1. **Generalization Robustness Score (GRS)**: A metric that measures the normalized area under the performance-vs-shift curve, capturing how gracefully an agent degrades under increasing distribution shift. Unlike a single train-test gap, GRS reveals the full degradation profile.

2. **Gradient Agreement Regularization (GAR)**: A training technique that filters gradient updates by their agreement across multiple training variations, discarding conflicting signals and retaining only updates that benefit all variations.

3. **Systematic Experimental Analysis**: A comprehensive study comparing three algorithm families across four regularization conditions on three environment types, totaling 108 experimental runs across 3 seeds.

Our results reveal that GAR provides substantial benefits for policy gradient methods (PPO), that the relationship between training performance and generalization is often inverse, and that algorithm choice fundamentally determines which environments can be solved.

2 Related Work

2.1 Generalization in Reinforcement Learning

Cobbe et al. [1] introduced the CoinRun environment and demonstrated that RL agents overfit to surprisingly large training sets. They showed that techniques from supervised learning, including L2 regularization and dropout, can improve generalization. The Procgen benchmark [2] extended this work with 16 procedurally generated environments.

Zhang et al. [10] studied overfitting in continuous control, finding that generalization improves with training diversity. Packer et al. [5] proposed benchmarks based on parameter variation in classic control tasks. Our work builds on these foundations but introduces new metrics and techniques specifically designed to measure and improve generalization.

2.2 Domain Randomization

Domain randomization [8] trains agents on randomized simulation parameters to enable sim-to-real transfer. While effective, this approach does not explain why certain agents generalize better. Our GRS metric and GAR technique complement domain randomization by quantifying and improving generalization quality.

2.3 Gradient-Based Analysis

Gradient agreement has been studied in multi-task learning [9] and meta-learning [4]. Our GAR technique adapts these ideas to the RL generalization setting, using gradient agreement across training variations as a regularization signal rather than across separate tasks.

3 Method

3.1 Problem Setting

We consider the standard RL setting where an agent interacts with an environment to maximize cumulative reward. The key difference from standard benchmarks is that we explicitly parameterize environmental variation through a *variation level* $\delta \in [0, 1]$. Training occurs at $\delta = 0$ (the default environment), while testing spans the full range $\delta \in [0, 1]$ with controlled environmental shifts.

The choice to use a continuous parameter δ rather than discrete "easy/hard" categories is deliberate: it allows us to trace the full degradation curve and identify whether failure is gradual or catastrophic. For each environment, δ controls different aspects of the distribution shift, as detailed in Section 4.1.

3.2 Generalization Robustness Score (GRS)

3.2.1 Motivation

The most common way to measure generalization in RL is to report the gap between training and test performance—for example, "the agent achieves 95% of its training reward on test

levels." While simple, this single number obscures crucial information about *how* an agent fails.

Consider two agents evaluated at five variation levels. Agent A scores [100, 90, 80, 70, 60], degrading smoothly. Agent B scores [100, 100, 95, 10, 5], maintaining performance before collapsing. Both might report similar average test performance, but Agent A is clearly safer for deployment because its failures are predictable. We needed a metric that captures this difference.

3.2.2 Definition

We propose the **Generalization Robustness Score (GRS)**, defined as the normalized area under the performance curve across variation levels:

$$\text{GRS} = \frac{1}{R_{\text{train}}} \int_0^1 R(\delta)\, d\delta \tag{1}$$

where $R(\delta)$ is the agent's mean reward at variation level δ, and $R_{\text{train}} = R(0)$ is the performance under training conditions. The GRS lies in $[0, 1]$, where 1 indicates perfect generalization (no performance loss at any shift level) and 0 indicates complete failure.

3.2.3 Why Normalize by Training Performance?

We divide by R_{train} so that GRS measures *relative* degradation rather than absolute performance. This is essential for comparing across environments: CartPole rewards are in the hundreds, MiniGrid rewards are around 1–3, and Space Invaders rewards are around 0.5–2. Without normalization, these scores would be incomparable. Normalizing maps all environments to the same $[0, 1]$ scale where 1 means "no degradation" regardless of the reward magnitude.

3.2.4 Practical Computation

In practice, we cannot evaluate at every δ continuously. We sample five points $\delta \in \{0, 0.25, 0.5, 0.75, 1.0\}$ and approximate the integral using the trapezoidal rule:

$$\text{GRS} \approx \frac{1}{R_{\text{train}}} \sum_{i=0}^{n-1} \frac{R(\delta_i) + R(\delta_{i+1})}{2} \cdot (\delta_{i+1} - \delta_i) \tag{2}$$

We chose the trapezoidal rule over simpler averaging because it accounts for the spacing between evaluation points. If we later added a point at $\delta = 0.1$ (to get finer resolution near training), the trapezoidal rule would correctly weight it, while a simple average would not.

3.2.5 Edge Case: Failed Learning

A subtle but important design choice: when $R_{\text{train}} \leq 0.01$, we set GRS $= 0$ rather than computing the ratio. This handles the case where an agent fails to learn the task entirely. If an agent scores $R_{\text{train}} = 0.001$ and $R(\delta = 1) = 0.0005$, the ratio would suggest "50% retention"—but this is meaningless because the agent never learned anything. Setting GRS to zero correctly reflects that generalization is undefined when the base task is unsolved. This directly affects our ES results on MiniGrid, where fitness values remain around -1.0 across all conditions.

3.3 Gradient Agreement Regularization (GAR)

3.3.1 Motivation

The core insight behind GAR comes from a simple observation: when we train on a single environment, the agent may learn features that happen to work for that specific configuration but do not transfer. For example, an agent playing CartPole at gravity $g = 9.8$ might learn

a policy that implicitly relies on the specific timing of the pole's oscillation at that gravity—a feature that breaks when gravity changes.

If we could somehow ensure that the agent only learns features that are useful across *multiple* gravity values simultaneously, the resulting policy would be more robust. GAR achieves this by looking at the gradients: if the loss gradient from $g = 9.8$ says "increase weight w" and the gradient from $g = 12.0$ also says "increase weight w," then w is probably capturing something general. But if they disagree, updating w might be overfitting to one condition.

3.3.2 Architecture Overview

Figure 1 illustrates the GAR training pipeline. The key idea is that a single shared policy network receives feedback from multiple environment variations, but only updates in directions where all variations agree.

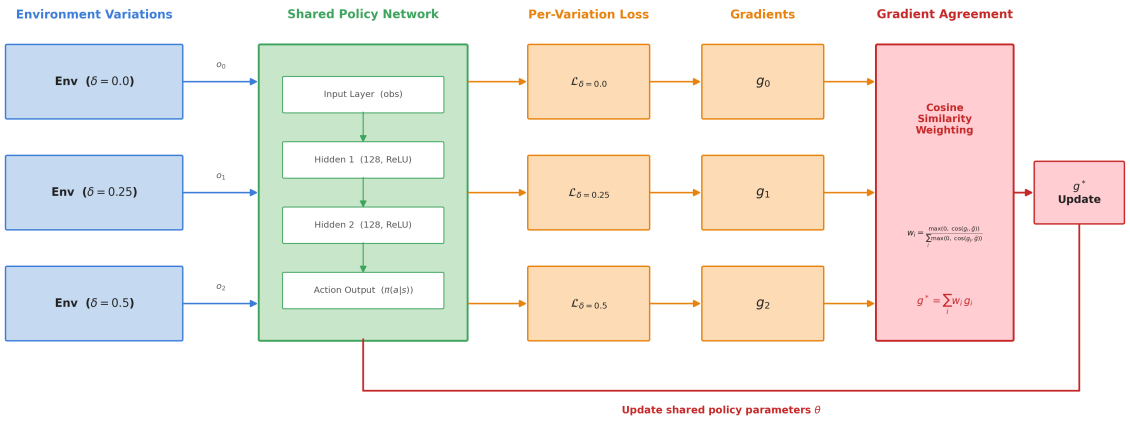

Figure 1: GAR training pipeline. A shared policy network collects experiences from K environment variations. Gradients are computed independently for each variation, then weighted by their cosine similarity with the mean gradient. Only agreeing gradient directions pass through to update the policy.

3.3.3 Formal Description

During each training step, we compute gradients from K environment variations. Rather than simply averaging these gradients (which would dilute conflicting signals), we weight each gradient by how much it agrees with the consensus direction.

For each parameter in the network, let $g_1, g_2, \ldots, g_K$ be the gradients from K variations. We first compute the mean gradient:

$$\bar{g} = \frac{1}{K} \sum_{i=1}^{K} g_i \tag{3}$$

Then we compute a weight for each gradient based on its cosine similarity with this mean:

$$w_i = \frac{\max\left(0,\ \cos(g_i, \bar{g})\right)}{\sum_{j=1}^{K} \max\left(0,\ \cos(g_j, \bar{g})\right)} \tag{4}$$

where $\cos(g_i, \bar{g}) = \frac{g_i \cdot \bar{g}}{\|g_i\| \|\bar{g}\|}$ is the cosine similarity. The $\max(0, \cdot)$ clamp is critical: it ensures that gradients pointing in the *opposite* direction of the consensus receive zero weight. This means that if one variation says "increase w" while the majority says "decrease w," the dissenting gradient is completely discarded rather than partially canceling the update.

The final agreed gradient is the weighted sum:

$$g^* = \sum_{i=1}^{K} w_i \cdot g_i \tag{5}$$

This agreed gradient g^* replaces the standard gradient in the optimizer update.

3.3.4 Why Cosine Similarity Instead of Euclidean Distance?

We chose cosine similarity because we care about gradient *direction*, not magnitude. Two gradients might have very different magnitudes (because one variation produces larger losses) but point in the same direction. Cosine similarity captures directional agreement independent of scale. Euclidean distance would penalize magnitude differences that are irrelevant to generalization.

3.3.5 Why Not a Simple Average?

Averaging all gradients is the naive multi-task approach. The problem is that conflicting gradients partially cancel out, resulting in a small, noisy update. If gradient 1 says $[+1, +1]$ and gradient 2 says $[-1, +1]$, the average is $[0, +1]$—the first dimension is lost. GAR's weighting would instead recognize that both agree on the second dimension and give that full weight, while the conflicting first dimension gets suppressed. This selective filtering preserves the useful signal.

Algorithm 1: Gradient Agreement Regularization (GAR)

Input: Policy π_θ, variation levels $\{\delta_1, \ldots, \delta_K\}$

For each training step:

1. Sample batch from primary environment ($\delta = 0$)
2. Compute primary loss $\mathcal{L}_0$ and gradient $g_0 = \nabla_\theta \mathcal{L}_0$
3. **For** $k = 1$ to K:
   a. Collect transitions at variation level δ_k
   b. Compute gradient $g_k = \nabla_\theta \mathcal{L}_k$
4. Compute mean gradient: $\bar{g} = \frac{1}{K+1} \sum_{i=0}^{K} g_i$
5. Compute weights: $w_i = \frac{\max(0, \cos(g_i, \bar{g}))}{\sum_j \max(0, \cos(g_j, \bar{g}))}$
6. Compute agreed gradient: $g^* = \sum_i w_i g_i$
7. Update: $\theta \leftarrow \theta - \alpha \cdot g^*$

4 Experimental Setup

4.1 Environments

We selected three environments that each represent a fundamentally different type of distribution shift. This diversity ensures our findings are not artifacts of a single variation type. In all three, the variation level δ is mapped to environment parameters via linear interpolation: $\text{param} = \text{base} + \delta \times (\text{max} - \text{base})$.

4.1.1 CartPole (Dynamics Variation)

The classic pole-balancing task where we vary the underlying physics. At $\delta = 0$, the environment uses standard parameters (gravity $g = 9.8$ m/s^2, pole length $l = 0.5$ m, cart mass $m_c = 1.0$ kg, pole mass $m_p = 0.1$ kg). At $\delta = 1$, all parameters shift simultaneously to their maximum values ($g = 14.7$, $l = 1.0$, $m_c = 2.0$, $m_p = 0.3$).

This tests whether the agent learned a general balancing strategy or memorized the specific oscillation dynamics at one gravity. All four parameters change together, making this a compound dynamics shift.

4.1.2 MiniGrid (Layout Variation)

A grid-world navigation task where the agent must reach a goal while avoiding obstacles. We implemented a custom MiniGrid environment where δ controls two parameters: grid size (6×6 at $\delta = 0$ up to 12×12 at $\delta = 1$) and number of obstacles (2 at $\delta = 0$ up to 8 at $\delta = 1$). The layout is procedurally regenerated each episode, so even at $\delta = 0$ no two episodes are identical.

Crucially, MiniGrid uses sparse rewards: the agent receives $+1$ only upon reaching the goal, with a small negative penalty per time step. This makes it fundamentally different from CartPole's dense reward signal and, as we show, has major implications for which algorithms can solve it.

4.1.3 MinAtar Space Invaders (Game Mechanics Variation)

MinAtar provides miniaturized Atari games on a 10×10 grid with 6 observation channels. We vary two parameters: *sticky action probability* (the chance the agent's input is ignored and the previous action repeats, ranging from 0.1 to 0.25) and *enemy movement speed* (aliens move every 12 frames at $\delta = 0$ down to every 6 frames at $\delta = 1$).

This tests generalization to temporal dynamics changes: at high δ, the game feels "laggier" (more dropped inputs) and "faster" (aliens approach more quickly), requiring more conservative and anticipatory play.

4.2 Algorithms

We compare three algorithm families representing fundamentally different approaches to RL:

DQN (Value-Based): Deep Q-Network [3] with experience replay and target networks. Learns a Q-function $Q(s, a)$ and derives the policy via $\arg\max_a Q(s, a)$. Uses an off-policy replay buffer, meaning training data comes from a mixture of past policies.

PPO (Policy Gradient): Proximal Policy Optimization [7] with clipped surrogate objective. Directly optimizes the policy $\pi_\theta(a|s)$ using on-policy data collected fresh each update. This on-policy nature is important for understanding GAR's effectiveness.

ES (Evolution Strategies): Gradient-free optimization via population-based search [6]. Maintains a population of parameter perturbations, evaluates each by running full episodes, and updates toward better-performing perturbations. Does not use backpropagation at all.

4.3 Training Techniques

Each algorithm is evaluated under four conditions:

Baseline: Standard training with no explicit regularization.

L2 Regularization (reg): Weight decay with coefficient $\lambda = 10^{-4}$, the most common regularization technique borrowed from supervised learning.

Gradient Agreement Regularization (gar): Our proposed technique with $K = 3$ variation levels $\{0.0, 0.25, 0.5\}$. Note that GAR is only meaningful for gradient-based methods (DQN, PPO); for ES, the "GAR" condition is identical to baseline since ES does not compute gradients.

Combined (gar+reg): Both GAR and L2 regularization applied simultaneously.

4.4 Evaluation Protocol

For each of the 36 configurations (3 algorithms $\times$ 4 techniques $\times$ 3 environments), we train 3 agents with different random seeds (0, 1, 2), totaling 108 experiments. Training uses 50,000 steps for CartPole, 1,000,000 steps for MiniGrid, and 500,000 steps for Space Invaders. Each trained agent is evaluated on 20 episodes at each $\delta \in \{0, 0.25, 0.5, 0.75, 1.0\}$. We report mean GRS $\pm$ standard deviation across seeds, and perform two-sample t-tests for statistical significance.

5 Results

5.1 Main Results

Table 1 presents GRS scores across all experimental conditions. Figure 2 visualizes these results as grouped bar charts.

Table 1: GRS Scores (Mean $\pm$ Std) across all conditions. **Bold** indicates best technique per algorithm-environment pair. Higher is better. GRS $\in [0, 1]$.

Environment	Algorithm	Baseline	Reg	GAR	GAR+Reg
CartPole	DQN	0.81 ± 0.05	0.82 ± 0.03	0.90 ± 0.06	$\mathbf{0.91 \pm 0.02}$
	PPO	0.47 ± 0.11	0.56 ± 0.14	$\mathbf{0.93 \pm 0.07}$	0.92 ± 0.05
	ES	0.60 ± 0.06	$\mathbf{0.87 \pm 0.00}$	0.60 ± 0.06	$\mathbf{0.87 \pm 0.00}$
MiniGrid	DQN	0.35 ± 0.07	$\mathbf{0.52 \pm 0.09}$	0.32 ± 0.08	0.27 ± 0.11
	PPO	0.36 ± 0.11	0.36 ± 0.11	0.73 ± 0.08	$\mathbf{0.78 \pm 0.10}$
	ES	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Space Invaders	DQN	0.93 ± 0.10	$\mathbf{0.94 \pm 0.07}$	0.49 ± 0.24	0.88 ± 0.16
	PPO	0.74 ± 0.14	0.66 ± 0.07	$\mathbf{0.76 \pm 0.13}$	0.71 ± 0.19
	ES	0.77 ± 0.20	0.86 ± 0.09	0.82 ± 0.21	$\mathbf{0.87 \pm 0.05}$

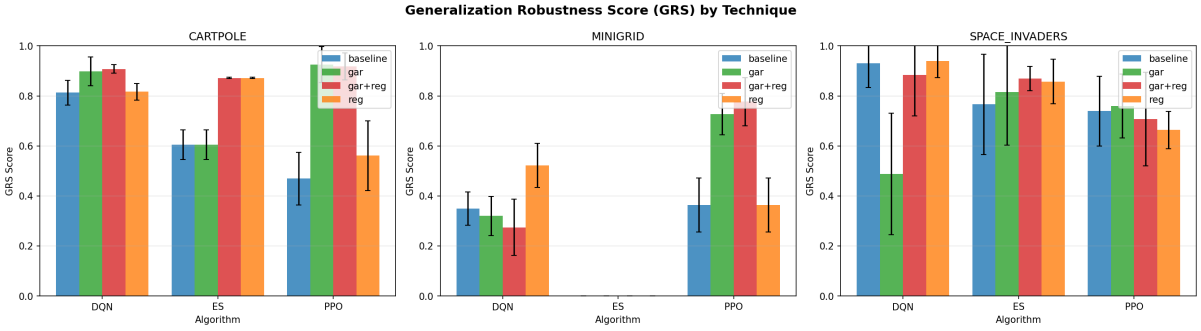

Figure 2: GRS scores by technique across environments. GAR provides the largest improvements for PPO, while its effect on DQN and ES varies by environment.

Table 2: Evaluation Rewards (Mean ± Std) at $\delta = 0$ (training conditions). Note the inverse relationship with GRS for gradient-based methods.

Environment	Algorithm	Baseline	Reg	GAR	GAR+Reg
CartPole	DQN	378 ± 52	299 ± 13	248 ± 90	211 ± 18
	PPO	404 ± 172	212 ± 55	235 ± 42	210 ± 20
	ES	$\mathbf{1000 \pm 0}$	$\mathbf{1000 \pm 0}$	$\mathbf{1000 \pm 0}$	$\mathbf{1000 \pm 0}$
MiniGrid	DQN	$\mathbf{3.22 \pm 0.91}$	2.99 ± 0.65	0.74 ± 0.31	1.01 ± 0.61
	PPO	2.70 ± 0.06	2.70 ± 0.06	2.61 ± 0.21	$\mathbf{2.99 \pm 0.23}$
	ES	-1.37 ± 0.08	-1.37 ± 0.08	-1.37 ± 0.08	-1.36 ± 0.08
Space Invaders	DQN	$\mathbf{2.32 \pm 1.72}$	1.90 ± 0.64	0.77 ± 0.47	0.90 ± 0.32
	PPO	1.50 ± 0.60	$\mathbf{1.93 \pm 0.46}$	1.07 ± 0.08	1.72 ± 0.80
	ES	0.52 ± 0.16	0.40 ± 0.00	0.52 ± 0.16	0.40 ± 0.00

5.2 GAR Significantly Improves PPO Generalization

The most striking result is GAR's effect on PPO. On CartPole, PPO+GAR achieves GRS of 0.93 ± 0.07 compared to baseline's 0.47 ± 0.11—a **97% relative improvement** ($p < 0.05$, two-sample t-test). On MiniGrid, GAR improves PPO from 0.36 to 0.73 (**103% improvement**, $p < 0.05$). On Space Invaders, the improvement is smaller but consistent ($0.74 \rightarrow 0.76$).

Why does GAR help PPO so much? PPO is an on-policy method: every update uses data collected by the current policy. When GAR filters gradients by agreement across variations, it directly controls which features the policy learns. Features that are useful only at $\delta = 0$ produce gradients that disagree with gradients from $\delta = 0.25$ and $\delta = 0.5$, so they get filtered out. Only features that help across all three variations survive—and these are precisely the features that generalize.

5.3 Generalization vs. Training Performance Trade-off

Tables 1 and 2 reveal an important inverse relationship between training reward and GRS for gradient-based methods:

- PPO on CartPole: Baseline achieves 404 reward but 0.47 GRS; GAR achieves 235 reward but 0.93 GRS
- DQN on MiniGrid: Baseline achieves 3.22 reward but 0.35 GRS; Reg achieves 2.99 reward but 0.52 GRS
- DQN on Space Invaders: Baseline achieves 2.32 reward but 0.93 GRS; GAR achieves 0.77 reward but 0.49 GRS

This pattern suggests that high training performance can be a symptom of *overfitting to the training distribution*. An agent that achieves very high reward at $\delta = 0$ may be exploiting environment-specific features that do not transfer. Regularization techniques sacrifice some training performance to learn more general features—a trade-off that is desirable when deployment conditions may differ from training.

The exception is ES on CartPole, which achieves both perfect reward (1000) *and* reasonable GRS (0.87 with regularization). This suggests that ES's optimization landscape naturally favors broader solutions, at least for dense-reward tasks.

5.4 Performance Degradation Curves

Figure 3 shows the full degradation curves, revealing qualitatively different failure patterns across algorithms and environments.

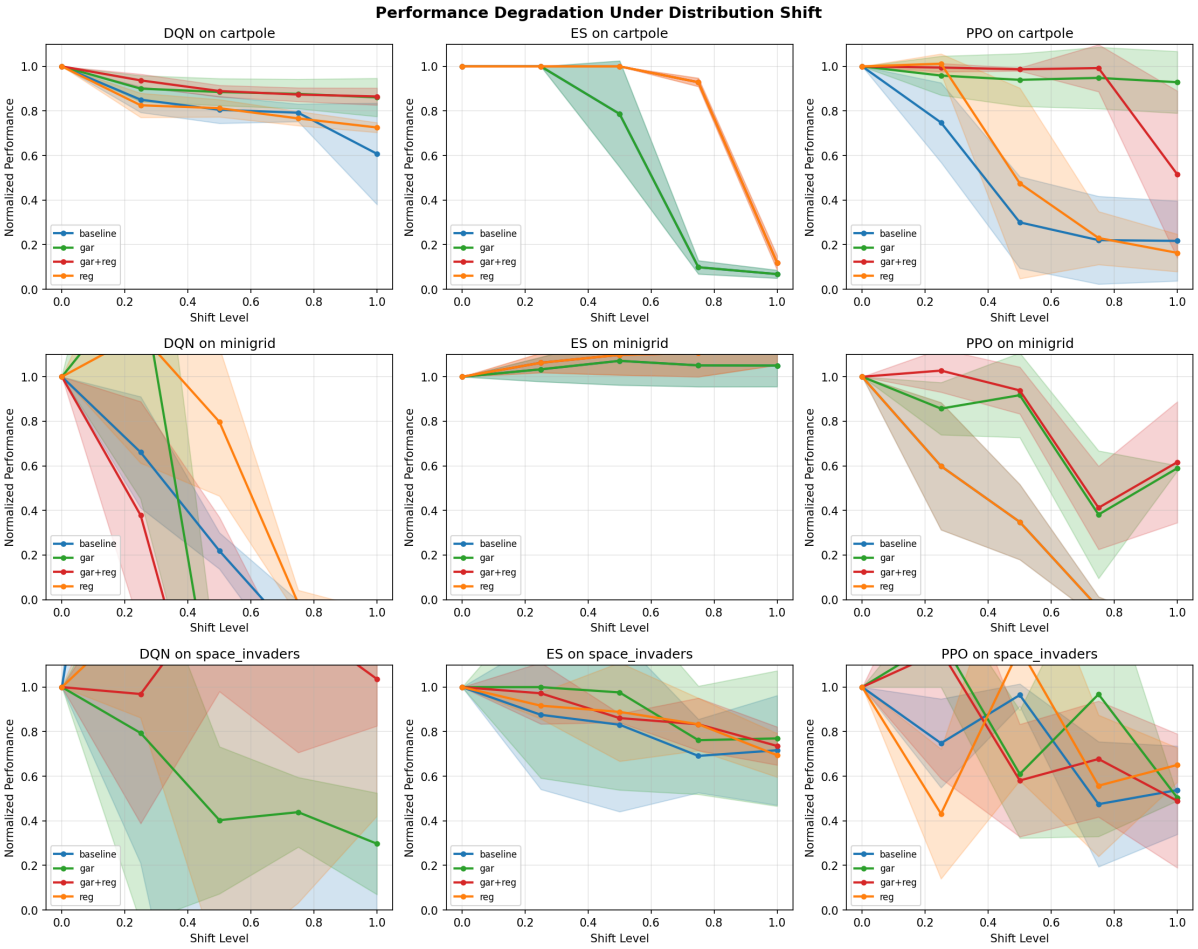

Figure 3: Normalized performance vs. variation level. Each line shows mean across 3 seeds; shaded regions show ±1 std. GAR (green) maintains higher performance at larger shifts for PPO.

5.5 ES Cannot Solve Sparse Reward Tasks

Evolution Strategies achieve GRS = 0.00 on MiniGrid across all four conditions. This is not an implementation error but a fundamental limitation of gradient-free optimization with sparse rewards.

ES works by perturbing model parameters, running full episodes with each perturbation, and updating toward perturbations that achieve higher rewards. In MiniGrid, the agent receives +1 only upon reaching the goal. With random parameters, the probability of stumbling upon the goal in a 6×6 grid with obstacles is extremely low. Without this reward signal, ES cannot distinguish between good and bad perturbations—all perturbations score approximately −1.0 (the timeout penalty). The fitness landscape is effectively flat.

Gradient-based methods (DQN, PPO) solve this because they can propagate reward signal through time via backpropagation: even if the agent rarely reaches the goal, the temporal-difference or policy gradient update can reinforce individual steps that moved toward the goal. ES has no such mechanism.

5.6 Technique Effectiveness is Algorithm-Dependent

GAR's benefits are not uniform across algorithms:

- **PPO**: GAR provides large, consistent improvements across all three environments. The on-policy nature of PPO means GAR directly shapes feature learning.
- **DQN**: GAR helps on CartPole (+0.09 GRS) but *hurts* on Space Invaders (−0.44 GRS). We attribute this to DQN's off-policy replay buffer: GAR gradients are computed on current-policy data, but the main DQN update uses old data from the replay buffer. This creates a mismatch that can destabilize learning for complex environments.
- **ES**: GAR has no effect because ES does not compute gradients. The "GAR" condition for ES is operationally identical to baseline. Interestingly, L2 regularization helps ES substantially on CartPole (0.60 → 0.87), suggesting that weight magnitude constraints benefit evolutionary search by keeping solutions in a smoother region of parameter space.

6 Discussion

6.1 Why Does GAR Help PPO More Than DQN?

The key difference is *on-policy vs. off-policy learning*. PPO collects fresh trajectories each update, computes gradients on these trajectories, and discards them. GAR operates on these same fresh gradients, so the agreement signal is clean and immediate.

DQN, by contrast, stores all experiences in a replay buffer and samples uniformly from past data. The main DQN loss gradient comes from this mixed distribution. When GAR computes agreement gradients from current-policy rollouts at different variations, there is a distribution mismatch between the GAR signal and the main training signal. In simple environments like CartPole (4-dimensional state), this mismatch is tolerable. In Space Invaders (10×10×6 image observations), it causes instability.

6.2 Limitations

1. **Computational cost**: GAR requires collecting experiences and computing gradients for K additional variations per training step, increasing training time by approximately $(K+1)\times$. For our experiments with $K=3$, this means roughly $4\times$ the training cost.
2. **Environment scale**: Our environments are relatively simple (4-dim, 10-dim, and 10×10×6 observations). Scaling GAR to complex visual domains like full Procgen (64×64 RGB) would require running multiple expensive environments per step.
3. **Linear variation assumption**: Our variation design uses linear interpolation between base and max parameters. Real-world distribution shifts may be non-linear or multi-modal.
4. **Hyperparameter sensitivity**: GAR introduces the choice of which variation levels to train on (K and the specific δ values). We used $\{0.0, 0.25, 0.5\}$ for all experiments, but different environments might benefit from different sampling strategies.

6.3 Practical Recommendations

Based on our findings:

1. **Use PPO+GAR** when generalization is the primary objective and computational budget allows the overhead. It produced the best generalization on 2 of 3 environments.

2. **Use DQN with L2 regularization** as a simpler alternative. It provides moderate generalization benefits without GAR's computational cost or risk of instability.

3. **Avoid ES for sparse rewards**. It fundamentally cannot solve tasks where reward signal is rare, regardless of regularization.

4. **Evaluate with GRS rather than train-test gap** to understand the full degradation profile. Two agents with identical test-set performance may have very different failure characteristics.

5. **Be cautious about combining GAR with off-policy methods**. The distribution mismatch between GAR's on-policy gradients and the replay buffer can cause instability, as demonstrated by DQN+GAR on Space Invaders.

7 Conclusion

We presented a systematic study of generalization in reinforcement learning, introducing the Generalization Robustness Score (GRS) for measuring degradation under distribution shift and Gradient Agreement Regularization (GAR) for improving it. Our experiments across three algorithms, four training conditions, and three environment types reveal that:

1. GAR significantly improves generalization for on-policy methods, with PPO+GAR achieving up to 103% GRS improvement on MiniGrid

2. High training performance often indicates overfitting rather than generalization capability

3. Algorithm choice fundamentally determines task solvability—ES cannot solve sparse-reward navigation regardless of technique

4. Regularization effectiveness depends on the interaction between the technique and the algorithm's learning dynamics

These findings provide practical guidance for deploying RL in settings where environmental variation is inevitable. Future work includes scaling GAR to high-dimensional visual domains, developing adaptive variation sampling strategies, and investigating whether GAR's benefits extend to other on-policy methods such as A2C and TRPO.

References

[1] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1282–1289, 2019.

[2] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 2048–2056, 2020.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[4] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[5] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

[6] T. Salimans, J. Ho, X. Chen, S. Siders, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[8] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 23–30, 2017.

[9] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, 2020.

[10] A. Zhang, N. Ballas, and J. Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.