DESIGN PRINCIPLES FOR ROBUST REINFORCEMENT LEARNING SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep reinforcement learning (RL) algorithms must concurrently coordinate exploration, policy improvement, and neural network optimization. While prior work has seen great success in optimizing these components in isolation, this paper argues that the asynchronous interactions between them are the primary driver of sensitivity and instability in modern deep RL systems. We demonstrate that well-known vulnerabilities, such as batch size limits in DQN, clipping bounds in PPO, and entropy tuning in continuous control, are often symptoms of *hyperparameter collusion*, where a sub-optimal choice in one component is required to mask inadequacies in another. Drawing an analogy to concurrent software systems, we propose a set of algorithm-agnostic design patterns aimed at ensuring the correctness of each component. We empirically show that these principles significantly improve the robustness and peak performance of widely-used deep RL agents, including PPO and SAC. We hope this work facilitates greater scientific rigor by simplifying standard baselines and streamlining the evaluation of new algorithms.

1 INTRODUCTION

Designing reinforcement learning (RL) algorithms that learn stably and reliably is notoriously challenging. Minor, seemingly innocuous choices which would robustly improve performance in supervised learning, such as increasing the batch size or optimization budget, can have catastrophic effects on the performance of an RL agent. Consequently, researchers and practitioners frequently rely on tribal folklore and ad-hoc heuristics to tune their algorithms, without a clear mechanistic understanding of why certain hyperparameter values are necessary. This stands in stark contrast to supervised deep learning, where scaling batch sizes and compute budgets predictably accelerates convergence and improves generalization.

In this work, we present a model of deep reinforcement learning algorithms as the orchestration of multiple subprocesses: updating the agent's experience bank D_t with new data (*exploration*), updating the local optimization objective $\mathcal{L}_t$ (*policy improvement*), and updating the neural network parameters θ_t towards the policy improvement direction (*optimization*). See Figure 1 for a visualization of this. Historically, the concept of a reinforcement learning *algorithm* has been largely synonymous with the choice of policy improvement objective $\mathcal{L}_t$, with the construction of $\mathcal{D}_t$ and the optimization of θ_t left as 'implementation details' (Huang et al., 2022). We argue that these implementation details should be treated as first-class citizens in determining the stability and success of an algorithm.

Interaction and interference between algorithm components is at the heart of many known pathologies in reinforcement learning systems, most famously demonstrated in Baird's counterexample (Baird, 1995), and the deadly triad (Van Hasselt et al., 2018), where the interaction of function approximation, off-policy sampling, and bootstrapped updates leads to divergent learning dynamics. Drawing an analogy to concurrent systems, we characterize the destructive interference between algorithm components in the familiar terms of *side effects*, where a change to one process influences the others, and *race dynamics*, where an algorithm's behavior heavily depends on the random ordering of events due to competitive dynamics between processes. These pathologies frequently force system designers into a state that we will refer to as *hyperparameter collusion*—an antipattern where sub-optimal choices in one component (e.g., restricted batch sizes) are required to compensate for inadequacies in another (e.g., insufficient exploration). After presenting a formalism for this

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

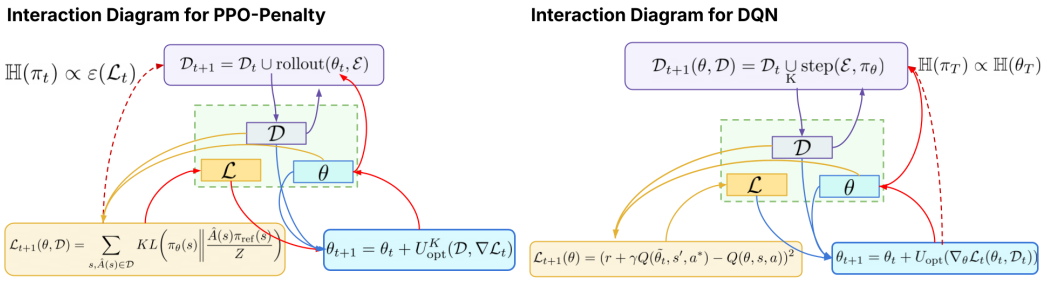


Figure 1: Instead of modeling RL algorithms as a sequential process acting on a learned policy via some unified global operator $\mathcal{T}$, it is useful to think about algorithm components as acting asynchronously on a shared state. Interactions which arise via the influence of each process on the shared state (solid red lines) result in side effects (dashed red lines). In this case, the entropy of the behaviour policy can be indirectly controlled by the stochasticity in the optimization process on θ_t, or by the policy improvement step regularization.

framework in Section 3.1, we apply it to explain several known instabilities in deep reinforcement learning algorithms.

We further draw on the language of concurrent systems to argue that reinforcement learning algorithms can be made significantly more performant and stable by validating the correctness of each process in isolation before attempting to tune the system as a whole. In particular, by ensuring the integrity of each component of an algorithm (for example, by validating that the sampling algorithm injects enough noise into the policy for exploration), it is possible both to reveal and mitigate hyperparameter collusion, and drastically reduce variance in final checkpoint performance. We propose a set of simple design principles alongside example implementations and validate their benefits empirically, finding significant improvements in hyperparameter sensitivity and variance under random seeds. Ultimately, we hope these insights will strengthen empirical rigor and improve reliability in the field of deep reinforcement learning. In summary, our core contributions are:

1. **A vocabulary for RL instabilities:** we formalize RL training as a set of asynchronous operators ($\mathcal{T}^{\text{opt}}, \mathcal{T}^{\text{pol}}, \mathcal{T}^{\text{exp}}$) and characterize their failure modes using the language of side effects and race dynamics.
2. **Diagnostic explanations of hyperparameter sensitivity:** we characterize the phenomenon of *hyperparameter collusion* and demonstrate that it provides a parsimonious explanation of well-known hyperparameter sensitivities.
3. **Algorithm-agnostic design patterns:** we provide a concrete recipe for ensuring the correctness and integrity of algorithm components, and empirically demonstrate that this recipe can improve performance, reduce variance, and mitigate hyperparameter sensitivity across popular baselines.

2 Background & Related Work

We assume a basic familiarity with the reinforcement learning problem; Appendix A.1 includes basic RL preliminaries and notation. The combination of neural networks with reinforcement learning algorithms has achieved success in a diverse array of domains (Tesauro, 1994; Silver et al., 2016; Berner et al., 2019; Brown & Sandholm, 2019; Bellemare et al., 2020), allowing the learner to leverage the structure (van der Pol et al., 2020) of the environment to accelerate learning (Du et al., 2019) and improve generalization (Kirk et al., 2023). Yet it also introduces additional risks in the form of value function divergence (Tsitsiklis & Van Roy, 1996; Van Hasselt et al., 2018), instability (Cetin et al., 2022), poor generalization (Di Langosco et al., 2022), and loss of plasticity (Lyle et al., 2023).

A great deal of recent work on deep reinforcement learning identifies algorithmic choices that improve the stability and convergence of the algorithm, for example by careful construction of bootstrap targets in data-efficient evaluation protocols (Kumar et al., 2021; Schwarzer et al., 2023), the use of target networks (Mnih et al., 2015), distributional targets (Bellemare et al., 2023; Farebrother

et al., 2024), normalization (Lyle et al., 2023; Lee et al., 2025; Salimans & Kingma, 2016), temporal abstraction (Park et al., 2025), custom architectures (Lee et al., 2024), and regularization (Kumar et al., 2023; Nauman et al., 2024). Policy gradient algorithms have benefited from the development of highly parallelized environments which do not penalize low data efficiency with greater wall clock time (Freeman et al., 2021; Lu et al., 2022; Rutherford et al., 2024), but still require some amount of regularization to prevent the learner from overfitting to each batch of rollouts, for example via trust regions (Schulman et al., 2017) or an entropy bonus (Haarnoja et al., 2018).

In spite of this progress, reinforcement learning algorithms remain notoriously difficult to benchmark (Henderson et al., 2018). At the core of this phenomenon lies fundamental tension: algorithmic choices which make an algorithm converge to an optimal policy faster and more data-efficiently, for example more aggressive policy improvement steps (Hernandez-Garcia & Sutton, 2019), faster-updated targets (Elsayed et al., 2024), and the reuse of stale data (D'Oro et al., 2022), tend to also amplify noise present in the learning process. As a result, evaluations of deep reinforcement learning algorithms which achieve competitive performance tend in aggregate to lack three critical properties. They are not *stable* (Bousquet & Elisseeff, 2002), they are not *reproducible* (Islam et al., 2017; Ahn et al., 2022), and they are not *replicable* (Eaton et al., 2023; Karbasi et al., 2023). While much discussion has gone into experimental protocols to increase the statistical robustness of performance comparisons of deep RL algorithms (Agarwal et al., 2021), this work takes a different perspective by exploring how the field's focus on data efficiency leads to algorithm design choices which make this estimation problem harder (Jordan et al., 2020).

3 RL TRAINING AS A CONCURRENT SYSTEM

Practical implementations of deep reinforcement learning algorithms specify exploration, policy improvement, and optimization as distinct, concurrent processes, each controlled by largely independent components of the codebase. The success of an implementation depends on its ability to both correctly specify each component as well as to coordinate their asynchronous progress. We propose to frame the dynamics of deep RL systems in the language of concurrency, characterizing two main categories of failure mode: *side effects*, where a change to one process influences the others, and *race dynamics*, where an algorithm's behaviour depends heavily on random orderings of events in different processes due to competitive dynamics between them. This framing will lay the groundwork for our explanations of sensitivities of existing systems, and will guide the design of more robust systems in later sections.

3.1 REINFORCEMENT LEARNING AS A DYNAMICAL SYSTEM

Reinforcement learning algorithms are naturally framed as dynamical systems. For example, Bellman iteration (Bellman, 1966), a workhorse of modern reinforcement learning, is characterized by the application of an operator $\mathcal{T}_{bell}$ to a predicted value function Q, yielding a discrete-time dynamical system

$$Q_{t+1}(s,a) = \mathcal{T}_{\text{Bell}} Q_t(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P_{env}(s'|s,a)}[\max_{a'} Q(s',a')] \; . \tag{1}$$

Prior works have analyzed the dynamics of the dynamical systems induced by this and similar operators under a variety of function approximation classes for Q (Tsitsiklis & Van Roy, 1996; Ghosh & Bellemare, 2020; Tang et al., 2023). This analysis typically decomposes the learner's state space and updates to it in order to facilitate analysis, for example expressing a neural network as the composition of a feature embedding Φ and a linear readout head with weights w. In this work, we extend this decomposition strategy to an even more inclusive notion of the learner's state. In addition to the current policy or value function parameters θ, we include objects such as the replay buffer $\mathcal{D}$ as well as the data required to compute the local optimization objective $\mathcal{L}_t$. Concretely, we will define the state $\mathcal{S}_t$ of a learning trajectory as the tuple $\mathcal{S}_t = (\theta_t, \mathcal{D}_t, \mathcal{L}_t)$.

In practice, the neural network optimization, policy improvement, and exploration components of an RL system are philosophically and computationally independent. For example, a Q-learning update can be used to produce a TD error minimization objective $\mathcal{L}_t^{TD}$ which is minimized by RMSProp or Adam, on data collected by epsilon-greedy, softmax, or count-based exploration strategies (or something else entirely). Thus, just as we decomposed the algorithm's state into policy parameters, local optimizer objective and associated data, and experience buffer, we will also decompose the

state update operator $\mathcal{T} : \mathcal{S}_t \to \mathcal{S}_{t+1}$ into independent components: the sampling/exploration operator which updates the data buffer $\mathcal{T}^{\text{exp}} : \mathcal{S}_t \to \mathcal{D}_{t+1}$, the policy improvement operator $\mathcal{T}^{\text{pol}} : \mathcal{S}_t \to \mathcal{L}_{t+1}$ which specifies the local optimization objective, and the network parameter optimizer update $\mathcal{T}^{\text{opt}} : \mathcal{S}_t \to \theta_{t+1}$.

This decomposition aligns naturally with modern distributed, asynchronous training setups such as A3C (Mnih et al., 2016), Impala (Espeholt et al., 2018), or more recent language model fine-tuning frameworks (Wu et al., 2025). In particular, it gives us the flexibility to model updates which occur at random times or with varying frequency, instead of viewing the dynamics of a reinforcement learning algorithm as a single operator $\mathcal{T} : \mathcal{S}_t \to \mathcal{S}_{t+1}$. An algorithm may interleave multiple data collection increments between optimizer steps, e.g. $\mathcal{T}^{\text{exp}} \circ \mathcal{T}^{\text{exp}} \circ \mathcal{T}^{\text{opt}} \circ \mathcal{T}^{\text{pol}}$. The exact ordering of these operations might vary between runs. This type of discrete stochasticity is poorly modeled by smooth, continuous-time systems with additive Gaussian noise; variance in an RL algorithm is more binary in nature, depending on binary events such as reaching a goal, receiving a reward, or visiting a particular state. We argue that deep RL training is thus best modeled as a concurrent program, where separate, asynchronous threads apply updates to a shared state, exposing the system to side effects and race conditions.

3.2 Characterizing instability

In Figure 1, we see that multi-hop causal chains allow design choices which ostensibly control one process, (e.g. optimizer batch size), to have side effects on others (e.g. exploration). We formalize this as a *side effects* problem, and discuss how side effects, along with competitive dynamics between subprocesses, can increase the sensitivity of an algorithm's performance on seemingly inocuous hyperparameter changes. Both pathologies can cause distinct algorithm components to *collude*, in the sense that suboptimality in one can cover for suboptimality in another.

3.2.1 Side effects

Although each operator $\mathcal{T}^i; i \in \{\text{opt}, \text{pol}, \text{exp}\}$ writes only to its respective component of the shared state, it takes the entire joint state as its input, and so its behaviour is sensitive to the effects of other operators on the state. For example, the optimizer will update the parameters used by the sampler, leading to potential *side effects* on performance via its indirect influence on exploration. In Figure 1, the influence of each update process on the others can be traced along a causal chain that passes through the shared state. This tight coupling is both necessary and beneficial in RL: policy updates rely on a constantly updating data buffer, and exploration depends on an improving policy. In a system with high *component integrity*, these interactions form a virtuous cycle where each operator strictly fulfills its explicit mathematical contract.

However, side effects can also obscure poor design choices by allowing one process to cover for another's deficiencies. Suppose that $\mathcal{T}^{\text{exp}}$ does not inject enough stochasticity to sample informative trajectories. This may go unnoticed if the stochasticity in the parameter sequence generated by the optimizer $\mathcal{T}^{\text{opt}}$ injects enough randomness into the sampling policy. This introduces confounding which can make the learning algorithm sensitive to other hyperparameter changes that affect the optimizer's stochasticity. Improvements which improve the signal-to-noise ratio in $\mathcal{T}^{\text{opt}}$ may in fact hurt performance in this case because they reveal an inadequacy in the exploration strategy. In this case, even if higher-signal optimizer steps would have helped under a sufficiently exploratory sampler, a practitioner may erroneously conclude that noise is playing some necessary regularizing effect on the optimization process. If second-order effects of a hyperparameter change dwarf its effect on the process it is ostensibly modifying, algorithms become brittle and increasingly difficult to refine or re-tune for new tasks.

3.2.2 Race dynamics

The operators we apply in an RL algorithm are non-commutative (e.g., $\mathcal{T}^{\text{opt}} \circ \mathcal{T}^{\text{exp}} \neq \mathcal{T}^{\text{exp}} \circ \mathcal{T}^{\text{opt}}$, as the parameters used by the sampler and data used by the optimizer will differ in depending on this ordering). This makes the system's learning dynamics highly sensitive to execution order, and to stochasticity in these operators' return values. Many sparse-reward environments, for example, exhibit binary returns as a function of whether or not the environment sampler stumbled upon a reward signal before the neural network collapsed into a deterministic, hard-to-update state.

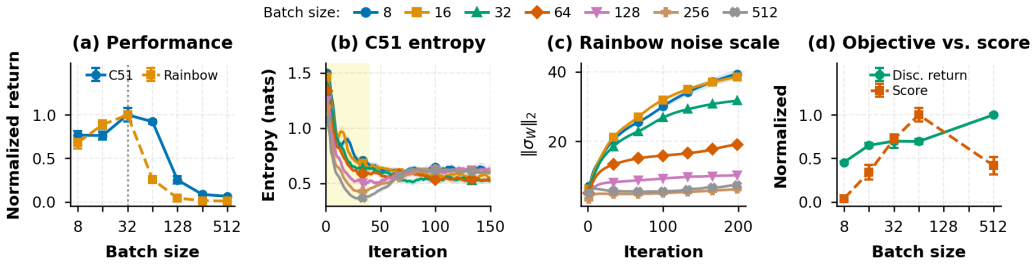


Figure 2: Atari agents tend to be extremely sensitive to batch size, exhibiting rapid performance degradation as it is increased. While C51 exhibits a clear early relationship between batch size and policy entropy, Rainbow is more complicated due to the secondary effect of batch size on the NoisyNets σ parameter. Finally, (d) shows that larger batch sizes are in fact effective at optimizing the *stated* objective used for training the Q-function.

A useful model for this type of return variability is that of hitting times in a Markov chain. In particular, the success of a training run will depend on the ordering of two hitting times: τ_{hit}, the hitting time for collecting sufficient informative transitions, and τ_{ext}, the entropy extinction time, at which point the policy entropy collapses, failing to generate or learn from informative samples. For the algorithm to have a hope of converging to an optimal policy, data collection must outpace policy collapse so that $\tau_{hit} < \tau_{ext}$ with high probability. This dynamic admits a direct analogue to race conditions in concurrent software: two different random seeds of the same algorithm in the same environment may produce wildly different final policies depending on the random orderings of these two events. This analogy is useful for developing some intuitions about the problem, though it is important to note that the 'race conditions' we consider in RL can occur even in entirely synchronous systems.

4 Explaining Instability with Our Formalism

In this section we examine the surprisingly long reach of two hyperparameter choices on an agent's learning dynamics: the batch size in Rainbow agents, and the entropy bonus in continuous control agents. That hyperparameter changes can have unpredictable effects in deep RL agents is common knowledge. This section demonstrates that the language of size effects and race conditions provides a parsimonious answer to several complex phenomena in empirical deep reinforcement learning.

4.1 Side Effects of Batch Size

Policy entropy. Value-based deep RL algorithms such as DQN (Mnih et al., 2015), C51 (Bellemare et al., 2017), and Rainbow (Hessel et al., 2018) are notoriously sensitive to batch size, often requiring shockingly small values (on the order of 16-64) to attain competitive performance. A working hypothesis in the literature (Ceron et al., 2023) is that smaller batches induce beneficial optimization stochasticity that drives exploration – in other words, the effect of batch size on performance is not driven by its direct effect on the optimization process $\mathcal{T}^{\text{opt}}$, but by its *side effect* on the data collection process $\mathcal{T}^{\text{exp}}$. We test this hypothesis by estimating the empirical greedy policy entropy (the empirical entropy of the argmax action on a batch of states obtained by uniformly sampling network outputs over 1000 contiguous optimizer steps) using the following estimator:

$$\widehat{H}_t = \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} \mathbb{H}_{\Theta \sim \mathcal{U}(\{\theta_t, \ldots, \theta_{t+1000}\})} \left(\text{argmax}_{a \in \mathcal{A}} Q_\Theta(s, a) \right). \tag{2}$$

Regularization on misspecified objectives. In the Arcade Learning Environment (ALE), agents are evaluated on undiscounted, unclipped cumulative game scores over long horizons (≈100k frames). The agent, however, trains on clipped rewards (≤ 1) discounted by $\gamma \in \{0.95, 0.99\}$ (depending on the algorithm, a different default γ may be used). Under this training objective, large rewards earned a few seconds after an action is taken contribute negligible value to the current state's performance

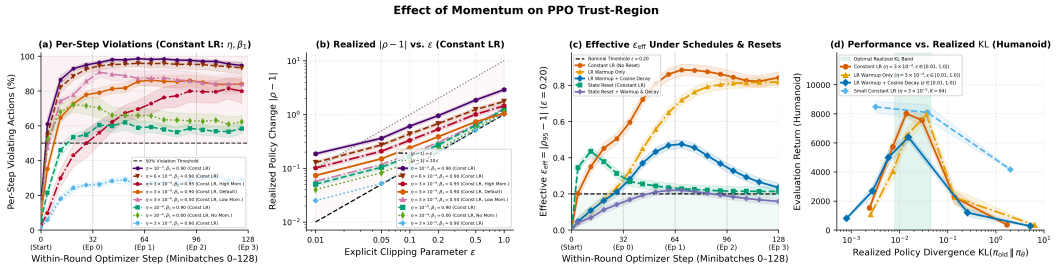


Figure 3: (a) We see that under a variety of clipping thresholds and optimizer hyperparameters, a large fraction of transitions are clipped even within the first epoch of training; (b) this translates to a much higher effective ε value than is explicitly set; (c) warming up the learning rate reduces the initial number of trust region violations, but not the final after four epochs; (d) policy updates of the same magnitude tend to produce similar results, independent of the hyperparameters used to achieve them.

estimate. We might therefore expect that learners which converge more quickly to local optima under this misspecified objective will perform worse than those which bounce around the loss landscape more. This reasoning is supported by Figure 2. When we evaluate C51 agents on the actual training objective (clipped, discounted return from the initial state), increasing the batch size even up to a value of 512 becomes monotonically beneficial.

4.2 Trust Region Violations and Optimizer Design

We next study the policy improvement step, turning our attention to trust region policy gradient methods, the most popular of which is Proximal Policy Optimization (Schulman et al., 2017), or PPO. In most PPO baseline implementations, the learner alternates between collecting data from the environment and running multiple epochs of optimization on this data. To avoid overfitting, each transition's gradients are clipped when they exceed a *trust region*, typically a factor of $1 \pm \varepsilon$ around the behaviour policy. The clipping parameter ε theoretically should give an approximate upper bound on the amount of deviation which can be achieved in a single policy improvement step. However, we show in Figure 3 that in fact this ostensible trust region is routinely violated by upwards of half of the actions in a training batch. We see that for small ε values, the average change in the policy under standard optimizer hyperparameters can span up to ten times the ε set in the learner's hyperparameters.

Adaptive optimizers have two features which can cause them to overshoot the PPO trust region. First, they do not automatically detect changepoints; when optimization starts on a new batch of data, the now-outdated statistics from the previous batch can lead to extremely large updates that quickly overshoot the trust region. Second, a non-zero momentum parameter can allow the policy to continue drifting in the direction of past gradients long after it has passed the trust region boundary, even though current gradients are zero. We see in subplot (c) that resetting optimizer statistics and using learning rate warmup and decay allow the optimizer to respect the trust region. Preventing trust region violations does not necessarily improve performance out of the box, as the trust region parameter itself will have been tuned to account for these violations in yet another example of hyperparameter collusion. Fixing the optimizer and then re-tuning ε, in contrast, maintains the integrity of the optimization and policy improvement process. By preventing the optimizer thread from overriding the safety bounds set by $\mathcal{T}^{\text{pol}}$, the hyperparameter tuning process becomes mathematically transparent rather than an exercise in managing collusion.

4.3 Race Dynamics Between Exploration and Exploitation

As an agent learns which actions are favorable, it concentrates probability mass on those actions. Consequently, when an on-policy agent uses its current policy to explore, its ability to discover novel behavior rapidly diminishes as learning progresses. This tension induces a *race dynamic*: if the agent does not visit states that provide sufficient information to escape a local optimum early, the likelihood that it ever will vanishes. This can lead to highly bimodal return distributions, particularly

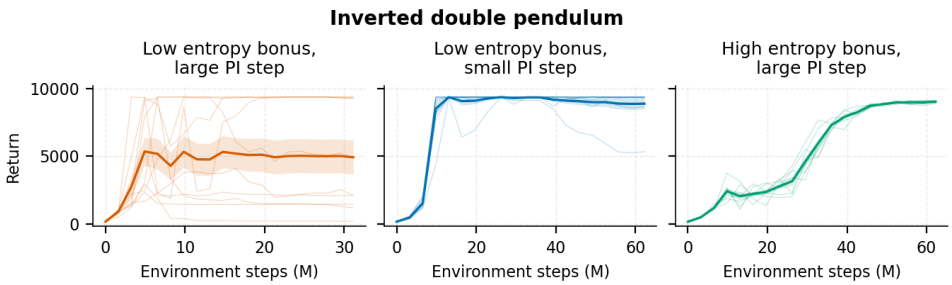


Figure 4: With an insufficient entropy bonus, PPO's performance on the humanoid environment converges to a roughly bimodal distribution, with half of runs converging to the optimal policy and the other half converging to a much lower performing policy. Reducing the size of the policy improvement steps earlier in training can reduce this, as can increasing the entropy of early policies.

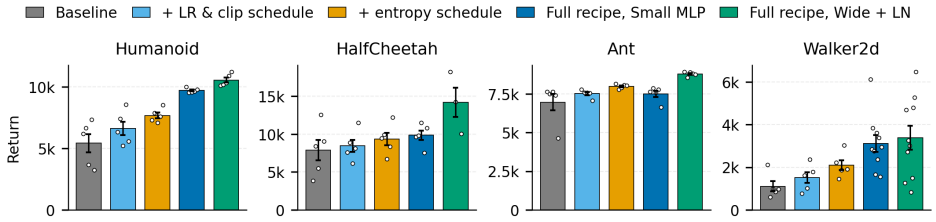


Figure 5: Additive effects of each step in recipe, consisting of implementing each design principle outlined in Section 5.1. We also evaluate on a larger architecture to assess the recipe's robustness.

in environments with many local optima in policy space, such as high-dimensional continuous control tasks. The key to mitigating such a race dynamic is to ensure that the policy is able to generate a large amount of diverse data *before* the learner is allowed to converge to a local optimum.

There are many ways to do this, of which we consider two in Figure 4. First, we can slow down the rate of policy change by reducing the permitted magnitude of each policy improvement step. Second, we can maintain the diversity of the behaviour policy for longer by explicitly decaying entropy over time, allowing the learner to follow an explore-then-exploit strategy. In both cases we do face a trade-off between rapidly converging to a local optimum vs slower improvement that more reliably reaches a near-global optimum. Unfortunately, many of the benchmarks we use to evaluate RL algorithms are rife with spurious local optima. For example, agents in HumanoidStandup may converge to a variety of strategies to stand, the highest-scoring of which involves acrobatically kicking the legs to 'jump' to the standing position. Avoiding the spurious attractor state of standing up without jumping requires careful tuning of the optimization process to ensure it explores sufficiently early on, while also allowing for large enough policy changes to latch onto the delicate kicking strategy once it is discovered.

5 Design Patterns

A recurring theme throughout our previous discussion is the notion that two wrongs do not make a right: setting excessively small batch sizes to make up for insufficient exploration algorithms, or overly-conservative clipping thresholds to accommodate optimizers that violate their constraints, introduces complexity into an already-uninterpretable system. In this section, we will present a few design principles which, as we will demonstrate, can improve the transparency of hyperparameter selection and the robustness of algorithm evaluations.

5.1 PATTERNS AND ANTIPATTERNS IN RL SYSTEMS DESIGN

We propose three design patterns, along with practical implementation guidance, towards ensuring that each operator ($\mathcal{T}^{\text{opt}}$, $\mathcal{T}^{\text{pol}}$, $\mathcal{T}^{\text{exp}}$) is well-specified, robustly fulfills its own mathematical objective, and can be scaled independently without breaking the system's asynchronous balance.

Pattern 1: high behavioural diversity in early phase of training. Insufficient exploration can drive difficult-to-diagnose hyperparameter sensitivities and high performance variance as suboptimal design choices for other subprocesses are required to cover for the inadequacy of the data collection process. We are agnostic to the specific exploration algorithm – whatever strategy is used, it should collect diverse data early in training, and allow policy convergence later in training. The flip side of this pattern is the hyperparameter collusion antipattern of co-opted stochasticity, whereby exploration is accidentally induced as a result of *optimizer* learning dynamics, destroying the integrity of both components.

Implementation: explicit exploration schedules. Rather than relying on small batch sizes to inject noise into the policy, agents should enforce exploration explicitly through an annealed entropy bonus. We observe that many algorithms benefit from a more exploratory phase early in training, but require lower policy entropy in later phases to converge to fine-grained, high-performing policies.

Pattern 2: calibrated policy improvement steps. The optimal magnitude of a policy improvement step should depend on the environment's sensitivity to policy changes. The objective given to the optimizer should therefore be calibrated to induce the desired magnitude of policy change. In practice, poorly-specified policy improvement objectives are often implicitly regularized by early-stopping the optimizer, meaning that larger optimizer budgets can rapidly degrade performance despite improving convergence on the local objective.

Implementation: smaller policy improvement steps on more data. Decaying the PPO clip magnitude during training, or increasing the target network EMA parameter in SAC, is a simple strategy which allows the policy improvement step's magnitude to adapt to increasing environment sensitivity. The success of these strategies depends on the stability of the optimizer. Collecting more data per policy improvement step tends to robustly improve performance and reduce variance across runs, at the cost of sample complexity.

Pattern 3: nonstationarity-aware optimizers. The fact that neural network optimization in deep reinforcement learning agents occurs on both a highly nonstationary data distribution and a highly nonstationary objective means that duplicating optimizer design choices from supervised learning is rarely an effective strategy. Optimizers borrowed from supervised learning typically assume a stationary or slowly-changing gradient stream, and behave erratically in highly nonstationary deep RL problems. At minimum, if gradient statistics are not relevant to the current optimization landscape, they should be discarded.

Implementation: optimizer state resets and warmup. We found previously that carrying stale gradient statistics across local objective updates is a primary driver of the trust region violations observed in Section 4.2. A simple, algorithm-agnostic design pattern to resolve this is to simply reset the optimizer statistics whenever the optimizer objective or data changes significantly, and then do a conservative learning rate warmup to refresh these statistics before the optimizer starts to take large steps again.

5.2 EMPIRICAL VALIDATION

The design patterns we enumerated do not necessarily require us to design entirely new algorithms from scratch. To a large extent, they can be implemented in many popular algorithms via thoughtful hyperparameter scheduling and optimizer design. We will illustrate this by implementing our recommendations in Soft Actor Critic Haarnoja et al. (2018) (SAC) and PPO (Schulman et al., 2017), two of the most popular baselines for continuous control tasks. We follow the implementation strategies described in the previous section, applying a standard recipe on top of the default Brax (Freeman et al., 2021) hyperparameters for each agent-environment pair.

Concretely, we schedule exploration over the course of training: in PPO the entropy bonus starts 20–50$\times$ above its default and decays to $\approx 5 \times 10^{-4}$ (exponential decay from 0.02 or 0.05, depending on the environment), and in SAC the target entropy is annealed linearly from $-0.05|\mathcal{A}|$ to $-c_{|\mathcal{A}|}|\mathcal{A}|$,

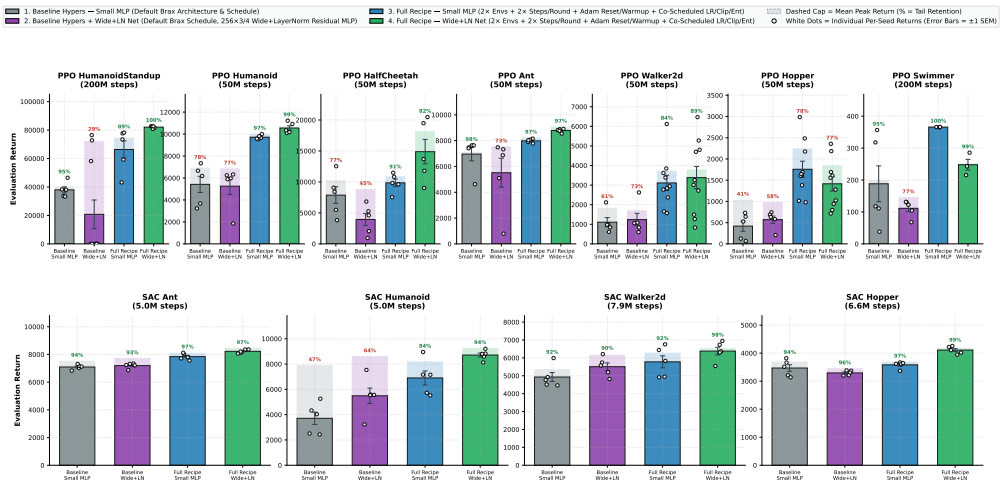


Figure 6: Following the three design principles listed in this section can reduce policy variability as well as increasing performance. While many environments still exhibit high variability in final performance, we observe fewer catastrophic learning failures.

where $c_{|\mathcal{A}|} = 0.5 \min\big(1 + (|\mathcal{A}| - 3)/7,\, 3\big)$ scales the final per-dimension target with the action dimension (from 0.5 for Hopper to 1.5 for Humanoid). To shrink the policy improvement step as training progresses, we linearly anneal the PPO clipping parameter from $\varepsilon = 0.2$ to 0.05 and the SAC target-network update rate from $\tau = 0.005$ to 0.001. In both algorithms the learning rate follows a linear warmup (10 steps) and then a cosine decay. For PPO, the peak learning rate decays by $4\times$ over training to match the reduction in ε. For SAC, it decays from 5×10^{-4} to 2.5×10^{-5}. In PPO, we reset the Adam moment estimates and step count at the start of every optimization round and re-warm the learning rate linearly over the first 64 gradient steps of each round. We also reduce the value-loss coefficient to 0.25 and double both the number of parallel environments and the number of gradient steps per round.

In Figure 6, we see that while the baseline exhibits high variability across seeds and sensitivity to the architecture width, incorporating our recipe reduces overall variability as well as increasing average performance, largely by reducing the number of low-performing outliers. In environments with a wider performance spread under the default hyperparameters, we see significant improvements in SAC, and we do not see regressions in environments where the algorithms are close to saturation, such as Ant.

6 Discussion

We conclude that instability is not a necessary component of reinforcement learning algorithms, provided one is willing to think hard about the problem being solved. There is not a free lunch, however, in that the way to make an RL algorithm stable usually requires making it less sample-efficient, especially at the beginning of training. The most sample-efficient algorithms will always lie on a knife's edge of instability, a place which may be acceptable for the purpose of generating a paper but is not a suitable place for trainees to operate.

On a more optimistic note, beyond these tradeoffs, this paper has highlighted several arbitrage opportunities as well. Following basic design principles concerning the isolation of side effects and correctly specifying learning objectives can dramatically improve hyperparameter interpretability and overall algorithm robustness. By shifting the community's focus away from managing hyperparameter collusion and towards restoring component integrity, we can begin to emulate the predictable scaling properties of supervised learning. When RL operators are functionally isolated and well-specified, increasing data collection and optimizer scaling strictly improves performance rather than collapsing the policy. Ultimately, we hope these design patterns provide a foundation for more robust, scalable, and legible reinforcement learning systems.

AI USE STATEMENT

AI was used primarily as a tool for drafting code, figures, and text during the development of this paper, in collaboration with the human authors. As models improved over the course of the writing of this paper, implementation was increasingly delegated to AI; as a result, the Atari experiments were largely written and analyzed by humans, whereas AI was used to implement several of the more involved Brax diagnostics and to track hyperparameter sweeps and ablations. While all of the ideas in this paper were generated by humans, AI was used to improve the manuscript's clarity.

ETHICS STATEMENT

This paper presents foundational research on the optimization dynamics and algorithmic stability of deep reinforcement learning systems. Our empirical evaluations rely entirely on standard, open-source physics simulations and game environments (Brax and the Arcade Learning Environment), which do not involve human subjects, personally identifiable data, or inherently harmful applications. By identifying and mitigating race conditions and hyperparameter collusion, our proposed design patterns aim to make reinforcement learning algorithms more predictable, stable, and reliable. While we recognize that general advancements in reinforcement learning can theoretically be deployed in dual-use scenarios, our specific contributions focus on improving the scientific rigor and safety of existing algorithms. To the best of our knowledge, this work does not introduce new concerns societal harms of AI, and we adhere fully to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our findings, we have thoroughly documented our experimental configurations and design patterns throughout the manuscript. The exact implementations of our explicit exploration schedules, calibrated policy improvement steps, and optimizer state resets are detailed in Section 5.2. Furthermore, we provide comprehensive ablation studies and hyperparameter robustness evaluations in Appendix A to isolate the individual impact of each proposed modification. All experiments were conducted using standard, widely available benchmarks including the Arcade Learning Environment and Brax. The complete source code, including our modified agent implementations and the scripts necessary to reproduce the training curves and diagnostic figures presented in this paper, is included in the anonymous supplementary materials.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Kwangjun Ahn, Prateek Jain, Ziwei Ji, Satyen Kale, Praneeth Netrapalli, and Gil I Shamir. Reproducibility in optimization: Theoretical framework and limits. *Advances in Neural Information Processing Systems*, 35:18022–18033, 2022.

Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, 1995. URL `https://api.semanticscholar.org/CorpusID:621595`.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pp. 449–458. Pmlr, 2017.

Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Marc G Bellemare, Will Dabney, and Mark Rowland. *Distributional reinforcement learning*. MIT Press, 2023.

Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Dkebiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar):499–526, 2002.

Justin A Boyan. Least-squares temporal difference learning. In *ICML*, pp. 49–56, 1999.

Steven Bradtke. Reinforcement learning applied to linear quadratic regulation. *Advances in neural information processing systems*, 5, 1992.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019.

Johan Samir Obando Ceron, Marc G Bellemare, and Pablo Samuel Castro. Small batch deep reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=wPqEvmwFEh.

Edoardo Cetin, Philip J Ball, Steve Roberts, and Oya Celiktutan. Stabilizing off-policy deep reinforcement learning from pixels. *arXiv preprint arXiv:2207.00986*, 2022.

Lauro Langosco Di Langosco, Jack Koch, Lee D Sharkey, Jacob Pfau, and David Krueger. Goal misgeneralization in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 12004–12019. PMLR, 2022.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Simon S Du, Sham M Kakade, Ruosong Wang, and Lin F Yang. Is a good representation sufficient for sample efficient reinforcement learning? *arXiv preprint arXiv:1910.03016*, 2019.

Eric Eaton, Marcel Hussing, Michael Kearns, and Jessica Sorrell. Replicable reinforcement learning. *Advances in Neural Information Processing Systems*, 36:15172–15185, 2023.

Mohamed Elsayed, Gautham Vasan, and A Rupam Mahmood. Streaming deep reinforcement learning finally works. *arXiv preprint arXiv:2410.14606*, 2024.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018. URL https://arxiv.org/abs/1802.01561.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint arXiv:2403.03950*, 2024.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

Dibya Ghosh and Marc G Bellemare. Representations for stable off-policy reinforcement learning. In *International Conference on Machine Learning*, pp. 3556–3565. PMLR, 2020.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

J Fernando Hernandez-Garcia and Richard S Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target. *arXiv preprint arXiv:1901.07510*, 2019.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *Reproducibility in Machine Learning Workshop (ICML)*, 2017.

Scott Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang, and Philip Thomas. Evaluating the performance of reinforcement learning algorithms. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4962–4973. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/jordan20a.html.

Amin Karbasi, Grigoris Velegkas, Lin Yang, and Felix Zhou. Replicability in reinforcement learning. *Advances in Neural Information Processing Systems*, 36:74702–74735, 2023.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. Dr3: Value-based deep reinforcement learning requires explicit regularization. *arXiv preprint arXiv:2112.04716*, 2021.

Saurabh Kumar, Henrik Marklund, and Benjamin Van Roy. Maintaining plasticity via regenerative regularization. *arXiv preprint arXiv:2308.11958*, 2023.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. *arXiv preprint arXiv:2410.09754*, 2024.

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning. *arXiv preprint arXiv:2502.15280*, 2025.

Chris Lu, Jakub Grudzien Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 2022.

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, pp. 23190–23211. PMLR, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. URL https://arxiv.org/abs/1602.01783.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control. *Advances in neural information processing systems*, 37:113038–113071, 2024.

Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes RL scalable. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=hguaupzLCU.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardhar Ingvarsson Juto, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, et al. Jaxmarl: Multi-agent rl environments and algorithms in jax. *Advances in Neural Information Processing Systems*, 37:50925–50951, 2024.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

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

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pp. 30365–30380. PMLR, 2023.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Richard S Sutton. Introduction to reinforcement learning with function approximation. In *Tutorial at the conference on neural information processing systems*, volume 33, 2015.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Yunhao Tang, Zhaohan Daniel Guo, Pierre Harvey Richemond, Bernardo Avila Pires, Yash Chandak, Remi Munos, Mark Rowland, Mohammad Gheshlaghi Azar, Charline Le Lan, Clare Lyle, András György, Shantanu Thakoor, Will Dabney, Bilal Piot, Daniele Calandriello, and Michal Valko. Understanding self-predictive learning for reinforcement learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 33632–33656. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/tang23d.html.

Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

John Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. *Advances in neural information processing systems*, 9, 1996.

Elise van der Pol, Daniel Worrall, Herke van Hoof, Frans Oliehoek, and Max Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 4199–4210. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/2be5f9c2e3620eb73c2972d7552b6cb5-Paper.pdf.

Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

Bo Wu, Sid Wang, Yunhao Tang, Jia Ding, Eryk Helenowski, Liang Tan, Tengyu Xu, Tushar Gowda, Zhengxing Chen, Chen Zhu, Xiaocheng Tang, Yundi Qian, Beibei Zhu, and Rui Hou. Llamarl: A distributed asynchronous reinforcement learning framework for efficient large-scale llm training, 2025. URL https://arxiv.org/abs/2505.24034.

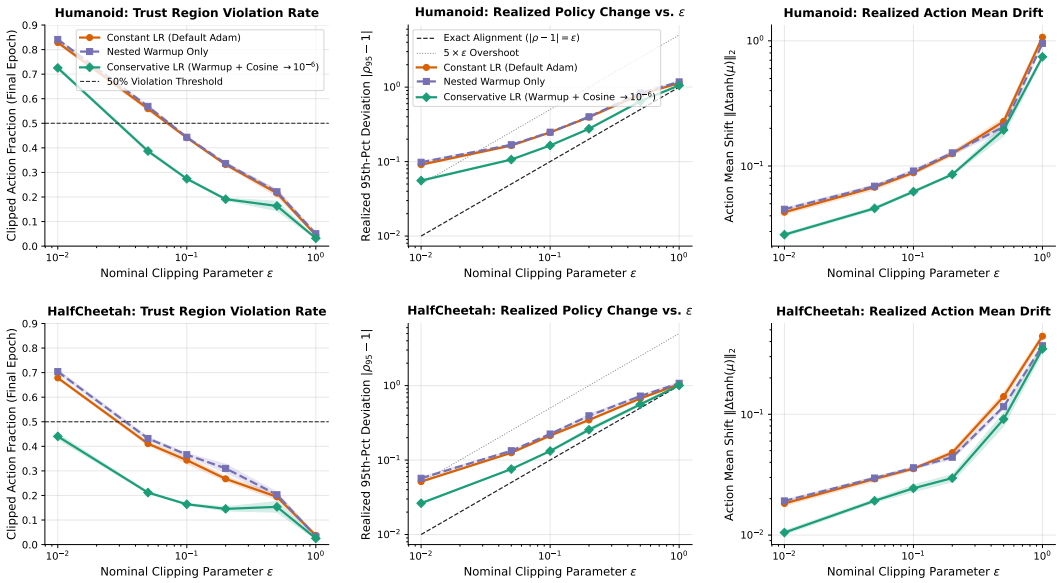


Figure 7: Humanoid and HalfCheetah trust region violation rate under different optimizer configurations.

A Appendix

A.1 Background on Reinforcement Learning

In reinforcement learning problems, an agent interacts with an environment $\mathcal{E}$ with the goal of maximizing some reward signal from that environment (Sutton, 2015). The environment is often modeled as a continuing Markov Decision Process (MDP) (Puterman, 1990). Algorithms for learning optimal policies in MDPs fall into two main classes: policy gradient methods (Sutton et al., 1999), which directly optimize policy parameters according to the gradient on the cost function

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,a\sim d^\pi}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)] \tag{3}$$

and temporal difference methods (Sutton, 1988), which aim to estimate the expected return of the optimal policy from each state-action pair, approximating the action of the Bellman operator

$$\mathcal{T}^\pi V(s) = \mathbb{E}_{a\sim\pi(s)}[R(s,a)] + \gamma \mathbb{E}_{s'\sim P(s,\pi(s))}[V(s)] \; . \tag{4}$$

This objective can then be translated into a differentiable loss for optimization by computing the difference between the current function output and the TD target (Bradtke, 1992; Boyan, 1999). Of critical importance to this work is the fact that both update rules involve the computation of an expectation which depends on the policy parameters.

A.2 Trust Region Overshooting Analysis

We provide supplemental analysis to our study of trust region violations in Section 4.2. In Figure 7, we expand on the analysis in the main paper, finding similar trends across clipping fraction, 'effective' ϵ induced by the degree of overshooting, and the magnitude of the change in the policy's mean. In Figure 8, we again compare different warmup and cooldown schedules on the inner optimization loop's learning rate, and find that across different optimizer configurations we end up seeing optimal performance at similar effective policy improvement step sizes, even though these occur at different clipping ε values for each configuration.

A.3 What causes baseline runs to collapse?

The Brax library tunes hyperparameters to a narrow 32-hidden-unit MLP. Although increasing model size will usually increase performance in a supervised learning problem, in our experiments

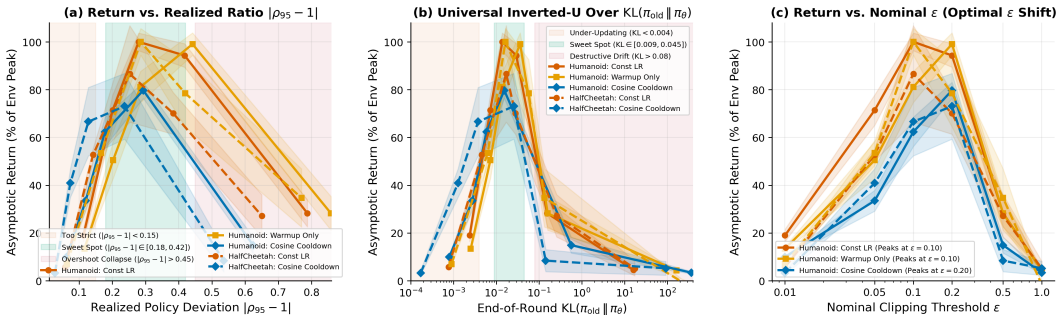


Figure 8: While clipping constraints and optimizer hyperparameters can influence the shape in addition to the magnitude of policy improvement steps, we see that in general similar-magnitude policy improvement steps translate to similar magnitude in performance as well.

this has the opposite effect. To illuminate the reason for this, it is helpful to take a closer look at the training dynamics induced by the default hyperparameters as well as our recipe. We plot four synchronized training time-series across all 7 PPO benchmarks (Part 1: high-dimensional locomotion HumanoidStandup, Humanoid, HalfCheetah, Ant; Part 2: planar cyclic locomotion Walker2d, Hopper, Swimmer) and all 4 SAC benchmarks (Ant, Humanoid, Walker2d, Hopper) in Figure 9. We plot four quantities: return, policy entropy, PPO clipping fraction / SAC temperature (though these might seem unrelated, we find that they are both symptoms of unstable neural network optimization), and gradient norms.

Return. Across PPO Humanoid, PPO Ant, PPO Walker2d, and PPO Hopper, the baseline-wide combination exhibits a characteristic "rise-then-collapse" signature, suggesting the original baseline is unable to take advantage of the larger model due to optimizer instability. On PPO HumanoidStandup the baseline frequently remains stuck at the ≈34K-point local optimum, while our recipe allows the agent to consistently break through into the higher-scoring policies. Our recipe exhibits consistent improvements on the SAC agents.

Policy entropy. Under constant entropy regularization (Baselines 1 & 2), Wide + LN policies either maintain excessively high action variance late in training (preventing fine-grained gait stabilization on Humanoid and Ant) or suffer sudden entropy jumps when large clipped updates destabilize policy log-std parameters. Under Full Recipe — Small MLP (blue) and Full Recipe — Wide + LN (green), co-scheduling produces a smooth, monotonic annealing trajectory—preserving wide exploration during early gait discovery (0–15M steps) while gradually tightening action precision in the final 20M steps without premature deterministic lock-in on Walker2d, Hopper, or Swimmer.

Clipping fraction and temperature. The clipping fraction measures the proportion of minibatch state-action pairs where the importance ratio exceeds $1 \pm \varepsilon$. In the baseline configuration, the clipping fraction surges to 0.25–0.38 during mid-to-late training on Humanoid, HalfCheetah, Ant, Walker2d, and Hopper—meaning over one-third of all gradient samples are actively hitting the trust-region boundary. Crucially, the surge in clipping fraction directly precedes or coincides with the onset of return degradation in Column 1. Under our recipe, co-scheduling LR decay with decay ($0.30 \rightarrow 0.10$ and $0.25 \rightarrow 0.15$) and resetting Adam statistics at the start of each iteration improves adherence to the trust region, bounding late-training clipping fraction to a more stable 0.06–0.14. On SAC using our recipe stabilizes automatic entropy temperature adaptation, preventing the oscillatory spikes that occur when critic target errors explode in the wide network with baseline hyperparameters.

Gradient norm. Column 4 provides support for the benefits of resetting optimizer statistics: in 2. Baseline — Wide + LN (coral), gradient norms escalate by 5x to 10x over the course of training (e.g., climbing from 0.8 to ¿ 6.5 on PPO Humanoid, ¿ 5.0 on PPO Ant, and ¿ 8.0 on SAC Humanoid). By zeroing Adam moments (μ, ν, count) at the beginning of each rollout iteration and applying a 10-minibatch linear LR warmup (Full Recipe, blue and green curves), gradient norms exhibit greater stability throughout training.

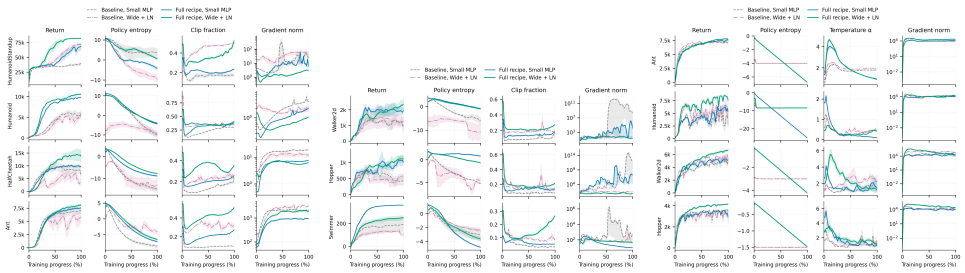
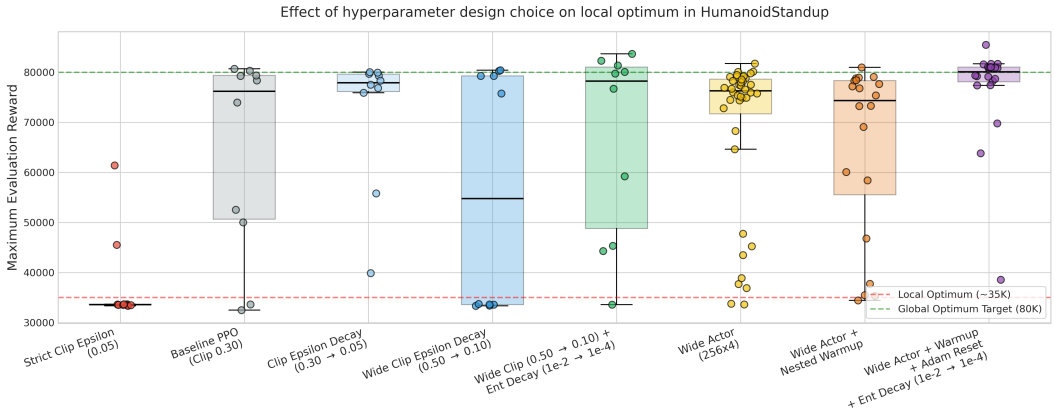


Figure 9: Detailed diagnostic information from PPO and SAC learning curves.



A.4 Recipe Summary

We provide a summary of our recipe components and the pathologies targeted by each in Table 1.

Table 1: Patterns and antipatterns in RL

Failure mode	**Observed antipattern**	**Implementation Fix**
Race condition: $\mathcal{T}^{\text{opt}}$ collapses policy before T^{exp} collects informative data	Shrinking batch size to co-opt optimizer noise for exploration	Maintain high policy entropy in early training to ensure $\tau_{hit} < \tau_{ext}$
Side effect: $\mathcal{T}^{\text{pol}}$ policy improvement steps too large	Early-stopping the optimizer on the local objective	Decay PPO clip ϵ or SAC EMA τ to prevent performance plateaus; increase data: PI magnitude ratio
Side effect: T^{opt} ignores T^{pol} limits using stale momentum cache	Tuning trust regions to absorb momentum overshoot	Flush optimizer state and warmup LR on new data

A.4.1 Details for 10

One surprising detail about the default hyperparameters for Walker2D and Hopper in the brax library is the relatively small discount factor ($\gamma = 0.97$) used. Because agents are evaluated on cumulative non-discounted episodic reward, discount factors often are used to trade-off between optimizing the real evaluation protocol against algorithmic stability considerations. Echoing our findings in Atari, we find here as well that a better-specified training objective can improve performance, provided other algorithm components are correctly calibrated. In Figure 10, we visualize our findings.

The figure is laid out as follows: the top row shows final-return bars for Walker2d (a) and Hopper (b): solid = $\gamma = 0.990$, hatched = $\gamma = 0.995$, ±1 SE. The bottom row shows mean learning curves

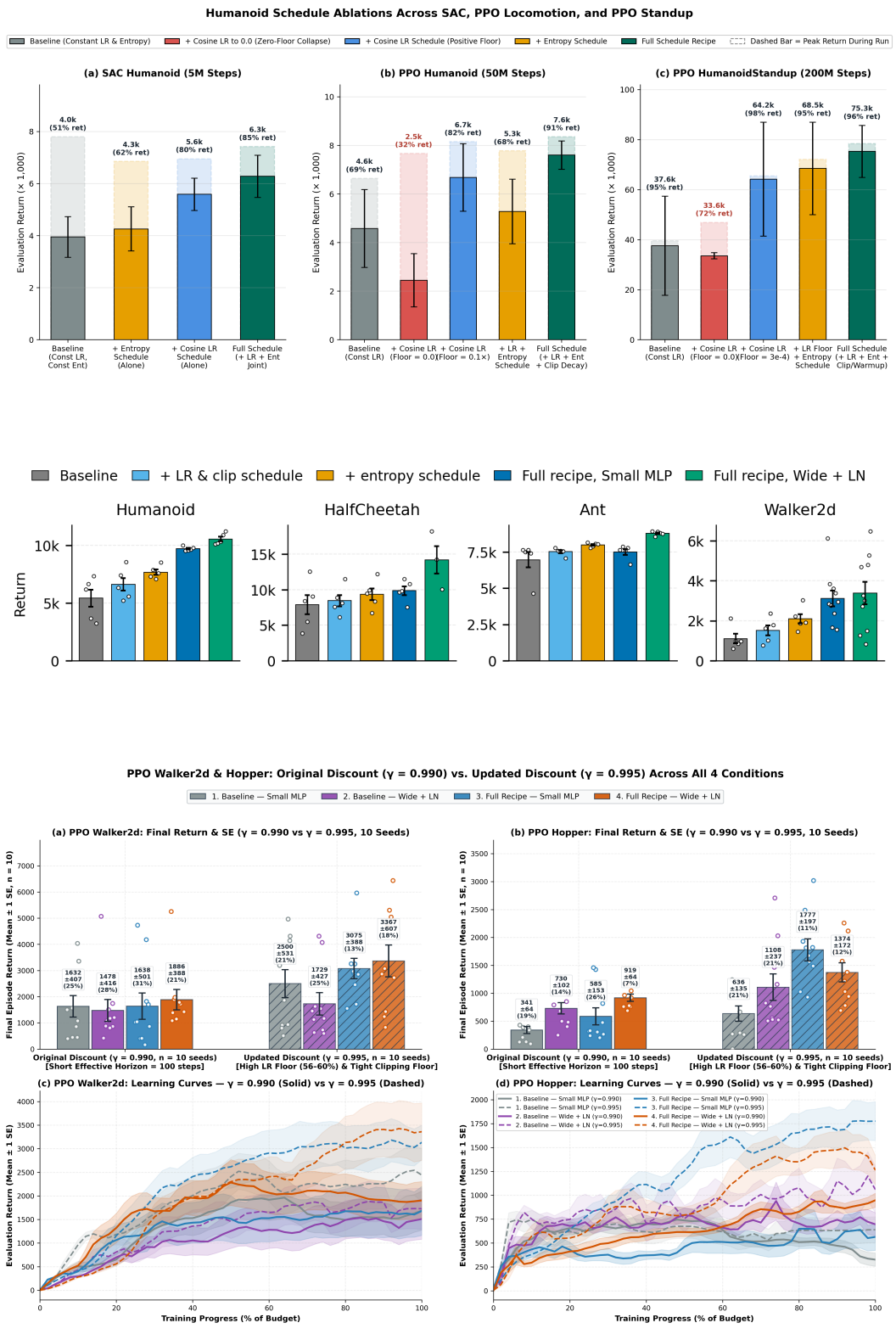


Figure 10: Effect of changing the discount γ on performance of both baseline and recipe variants of Brax agents.

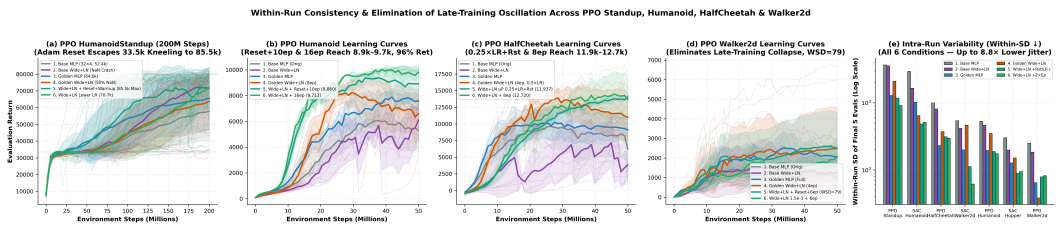


Figure 11: Within-trajectory consistency is improved by our recipe, as evidenced by reduced variation between evaluations.

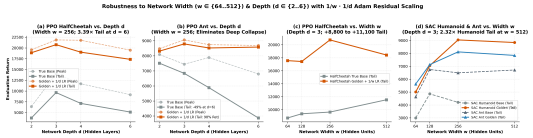


Figure 12: **Robustness to network architecture scaling.** Evaluation returns across HalfCheetah, Ant, and Humanoid as network depth (d) and width (w) increase. Under the baseline hyperparameters, increasing depth beyond $d = 2$ or $d = 3$ causes severe performance degradation, indicative of hyperparameter collusion tuned exclusively for small networks. By restoring component integrity, our recipe allows returns to remain stable or improve as architecture size scales.

(c, d): solid = 0.990, dashed = 0.995. We evaluate four conditions and 10 seeds in each environment. $\gamma = 0.995$ is paired with a higher late-training LR/clip floor to ensure that the network is able to adapt to changes in the input normalization statistics. We report final return as the mean over the final three evals of each seed. We see that in almost all cases, increasing γ improves performance, although the base hyperparameters with the wide MLP exhibit some instability; we also see larger gains from our recipe with the longer horizon as opposed to with the baseline hypers. This supports our overall claim that, provided the algorithm components are stable, correctly specifying the optimization objective will lead to better performance.

A.4.2 Effect of recipe on consistency

We next provide supporting evidence that maintaining component integrity can reduce *within-run* variance, in addition to cross-hyperparameter and cross-seed variance. In figure 11 subplots (a) to (d) we plot mean learning curves for PPO HumanoidStandup, Humanoid, HalfCheetah and Walker2d, for up to 6 conditions: base narrow MLP, base Wide+LN, recipe narrow MLP, recipe wide MLP, wide MLP with optimizer resets or learning rate decay, and wide MLP with 2x epochs. Subplot (e) shows within-run standard deviation over time for the 6 conditions. The reset/warmup and extra-epoch variants of Wide+LN reach the highest tail returns on Humanoid (8.9–9.7k) and HalfCheetah (13.8–14.2k). On Humanoid, HalfCheetah and Walker2d the baselines keep only 45–78% of their peak. On HumanoidStandup all Wide+LN variants reach roughly 62–72k. The spread between conditions is small compared with the seed-to-seed spread in the Golden runs.

Robustness to Network Width & Depth

We next do a deeper dive into the robustness of our recipe to variations in the network architecture, scaling depth and width in the HalfCheetah, Ant, and Humanoid environments. Under the baseline hyperparameters, going deeper hurts performance. HalfCheetah and Ant end-of-training returns drop quickly after $d = 3$. With our recipe along with standard learning rate scaling rules (1/d and 1/w for depth and width respectively), returns stay roughly flat with depth and width.

A.5 Hyperparameter robustness

A hallmark of hyperparameter collusion is extreme sensitivity to minor configuration changes, as the fragile balance between overlapping side-effects is easily disrupted. In Figure 13, we demonstrate that by maintaining the integrity of each algorithm component, our recipe significantly expands the

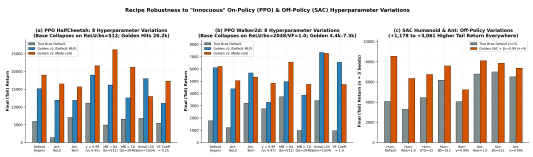


Figure 13: **Robustness to hyperparameter perturbation.** Final performance distributions across a sweep of hyperparameter perturbations (e.g., learning rate, entropy coefficient) around the baseline defaults. Under the baseline configuration, components rely on hyperparameter collusion to maintain stability, making the system highly brittle to minor perturbations. By enforcing component integrity through our design patterns, the agents decouple these asynchronous processes and maintain stable, high-performing policies across a significantly wider volume of the hyperparameter space.

stable hyperparameter volume. Agents maintain high performance across a wide grid-search of perturbations that otherwise cause the baseline to collapse.

A.5.1 Detailed step-by-step ablation

In Figure 14, we plot performance on all environments as we iteratively add recipe components. On top of the standard components (stable optimizer hyperparameters, policy improvement step decay, and early behaviour diversity), we also add a slightly improved architecture with a width of 256 and layer normalization included prior to nonlinearities, illustrating that the stable recipe allows architectural improvements to immediately translate to better performance (in contrast to the base hyperparameters, which often require additional tuning under the new architecture). For PPO, we add components in the following order:

1. Base
2. +LR & Clip schedule
3. +Entropy schedule (= Golden Small MLP, 'sm')
4. Full Recipe Small MLP (2× data + Adam reset/warmup)
5. Full Recipe Wide+LN

and for SAC:

1. Base
2. +LR & fast τ
3. Full Recipe Small MLP
4. +Wide+LN (Schedule v1)
5. Full Recipe Wide+LN

We see near-uniform benefits from each step across tasks, with the exception of Ant (which is already very close to optimal performance in the Base model). The relative gain from each step varies by environment: steps 2–3 (the schedule) give most of the retention gain on PPO Humanoid, while the 2× data + reset/warmup step (4) gives the biggest jump on Humanoid and Walker2d. Increasing the network architecture gives the biggest jump on HalfCheetah (+4.3k).

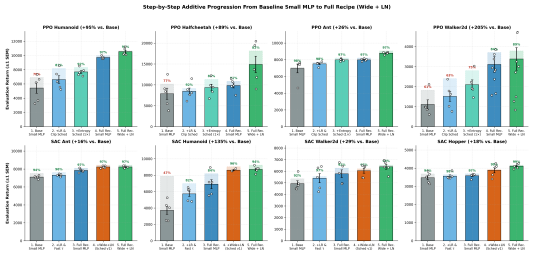


Figure 14: **Step-by-step additive ablation of recipe components.** Final performance across PPO and SAC environments as design patterns are iteratively applied. Adding calibrated policy improvement steps (+LR & clip schedule), explicit exploration (+entropy schedule), and optimizer state resets (Full Recipe) yields compounding, near-uniform benefits across tasks. Crucially, enforcing component integrity ensures these algorithmic improvements translate cleanly to larger architectures (Wide+LN) without requiring environment-specific retuning.