ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

«ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΦΑΡΜΟΓΕΣ»

«MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS”

Generalization and Robustness of RL Agents in Procedurally Generated Game Environments

Αναγνώστου Σταύρος

Ιούλιος 2026

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΦΑΡΜΟΓΕΣ»

«Master of Science in Artificial Intelligence and Applications”

Generalization and Robustness of RL Agents in Procedurally Generated Game Environments

Διπλωματική Εργασία που υποβλήθηκε στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Θεσσαλίας ως μέρος των απαιτήσεων για την απόκτηση Διπλώματος Μεταπτυχιακών Σπουδών Τεχνητή Νοημοσύνη και Εφαρμογές από τον/την

Αναγνώστου Σταύρος

Δήλωση Αυθεντικότητας, ζητήματα Copyright

«Ο/Η μεταπτυχιακός/ή φοιτητής/τρια που εκπόνησε την παρούσα διπλωματική εργασία φέρει ολόκληρη την ευθύνη προσδιορισμού της δίκαιης χρήσης του υλικού, η οποία ορίζεται στη βάση των εξής παραγόντων: του σκοπού και χαρακτήρα της χρήσης (μη-εμπορικός, μη-κερδοσκοπικός, αλλά εκπαιδευτικός-ερευνητικός), της φύσης του υλικού που χρησιμοποιεί (τμήμα του κειμένου, πίνακες, σχήματα, εικόνες κ.λπ.), του ποσοστού και της σημαντικότητας του τμήματος που χρησιμοποιεί σε σχέση με το όλο κείμενο υπό copyright, και των πιθανών συνεπειών της χρήσης αυτής στην αγορά ή την γενικότερη αξία του υπό copyright κειμένου».

Ο/Η Μεταπτυχιακός/ή Φοιτητής/τρια

Υπογραφή

Ιούλιος 2026

Τριμελής Εξεταστική Επιτροπή

Η παρούσα Μ.Δ.Ε. εγκρίθηκε ομόφωνα από την τριμελή εξεταστική επιτροπή η οποία ορίστηκε από την Σ.Ε. / Συνέλευση του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Θεσσαλίας, σύμφωνα με το νόμο και τον εγκεκριμένο Οδηγό Σπουδών του Π.Μ.Σ. «Τεχνητή Νοημοσύνη και Εφαρμογές». Τα μέλη της Επιτροπής ήταν:

1ο Μέλος (Επιβλέπων):	
2ο Μέλος:	
3ο Μέλος:	

Η έγκριση της Μ.Δ.Ε. από το Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Θεσσαλίας, δεν υποδηλώνει αποδοχή των απόψεων του συγγραφέα.

Περίληψη

Οι πράκτορες βαθιάς ενισχυτικής μάθησης τείνουν στην υπερπροσαρμογή: ένας πράκτορας που κατακτά τα συγκεκριμένα επίπεδα στα οποία εκπαιδεύτηκε συχνά αποτυγχάνει σε άγνωστα επίπεδα που προέρχονται από την ίδια κατανομή. Η παρούσα διπλωματική εργασία μελετά άμεσα αυτή την αστοχία, χρησιμοποιώντας το σύνολο αναφοράς Procgen, όπου ένας πράκτορας εκπαιδεύεται σε ένα σταθερό σύνολο 200 διαδικαστικά παραγόμενων επιπέδων και αξιολογείται στην πλήρη, μη φραγμένη κατανομή επιπέδων που δεν συνάντησε ποτέ. Η διαφορά μεταξύ της επίδοσης στην εκπαίδευση και στον έλεγχο, το λεγόμενο χάσμα γενίκευσης,

Η παρούσα εργασία συγκρίνει τρεις οικογένειες προσεγγίσεων με μια καλά ρυθμισμένη βάση PPO: την προσαρμοστική επαύξηση δεδομένων (UCB-DrAC)· έναν συνδυασμό επανάληψης επιπέδων κατά προτεραιότητα με επαύξηση (PLR + DrAC)· και μια εκτός πολιτικής, αξιακή βάση σύγκρισης (DQN). Τα αποτελέσματα συναθροίζονται σε έξι παιχνίδια με τον ενδοτεταρτημοριακό μέσο (IQM) και διαστήματα εμπιστοσύνης μέσω στρωματοποιημένου bootstrap, ακολουθώντας τις πρόσφατες βέλτιστες πρακτικές για την αξιολόγηση ενισχυτικής μάθησης με λίγους σπόρους.

Προκύπτουν τρία βασικά ευρήματα. Η προσαρμοστική επαύξηση είναι ο πιο αξιόπιστος μοχλός για το κλείσιμο του χάσματος, με το UCB-DrAC να επιτυγχάνει την υψηλότερη συνολική επίδοση (IQM 0,927 έναντι 0,879 της βάσης)· ωστόσο, το διάστημα εμπιστοσύνης του επικαλύπτεται με εκείνο της βάσης, επομένως η βελτίωση, αν και θετική ως σημειακή εκτίμηση, δεν είναι στατιστικά καθοριστική με τρεις σπόρους και εξαρτάται έντονα από το εκάστοτε παιχνίδι. Ο συνδυασμός PLR + DrAC παράγει το μικρότερο και πιο σταθερό χάσμα γενίκευσης από κάθε άλλη μέθοδο, αλλά με κόστος χαμηλότερες απόλυτες αποδόσεις, αναδεικνύοντας μια αντιστάθμιση μεταξύ του κλεισίματος του χάσματος και της μεγιστοποίησης της κορυφαίας επίδοσης. Το DQN αποδίδει κοντά σε μια τυχαία πολιτική μετά την κανονικοποίηση, επιβεβαιώνοντας ότι η εκτός πολιτικής επανάληψη εμπειρίας είναι ακατάλληλη για αυτό το πλαίσιο. Η συνεισφορά της εργασίας έγκειται σε μια ενιαία, ελεγχόμενη σύγκριση αυτών των μεθόδων υπό έναν κοινό αγωγό εκπαίδευσης, έναν κοινό προϋπολογισμό υπερπαραμέτρων και μια κοινή στατιστική μεθοδολογία, παρουσιαζόμενη με διαφάνεια ως προς την αβεβαιότητά της και όχι ως απλές σημειακές εκτιμήσεις.

Abstract

Deep reinforcement learning agents are prone to overfitting. An agent that masters the specific levels it was trained on often fails on unseen levels drawn from the same distribution. This thesis studies that failure directly, using the Procgen benchmark, in which an agent is trained on a fixed set of 200 procedurally generated levels and evaluated on the unbounded held-out distribution. The difference between training and test performance, termed the generalization gap, is taken as the central object of study.

This thesis compares three families of approach against a tuned PPO baseline: adaptive data augmentation (UCB-DrAC); a curriculum of prioritized level replay combined with augmentation (PLR + DrAC); and an off-policy, value-based contrast (DQN). Results are aggregated across six games using the interquartile mean (IQM) with stratified bootstrap confidence intervals, following recent best practice for low-seed reinforcement-learning evaluation.

Three findings emerge. Adaptive augmentation is the most reliable lever for closing the gap, with UCB-DrAC attaining the highest aggregate score (IQM 0.927 against 0.879 for the baseline); however, its confidence interval overlaps that of the baseline, so the improvement, though positive on the point estimate, is not statistically decisive at three seeds and is strongly game-dependent. Curriculum-based PLR + DrAC produces the smallest and most consistent generalization gap of any method, but at the cost of lower absolute returns, exposing a trade-off between closing the gap and maximizing peak performance. DQN performs close to a random policy after normalization, confirming that off-policy replay is poorly suited to this regime. The contribution of this work is a single, controlled comparison of these methods under one training pipeline, one hyperparameter budget and one statistical methodology, reported transparently with its uncertainty rather than as bare point estimates

Πίνακας Περιεχομένων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Chapter 1: Introduction

1.1 Background and Motivation

Reinforcement learning (RL) has produced agents that exceed human performance in domains once thought to require uniquely human judgment: Atari games, the board game Go, real-time strategy games such as StarCraft II, and increasingly complex robotic control tasks (e.g. Mnih et al., 2015; Silver et al., 2016; Vinyals et al., 2019). These results share a common evaluation protocol: an agent is trained and tested in the same environment, often on the exact same level, map, or initial configuration. The agent is never asked to do anything it has not, in some sense, already practiced.

This protocol conceals a basic question: has the agent learned a strategy, or has it merely memorized a sequence of actions that happens to work for one specific environment instantiation? In supervised learning this distinction is foundational. No credible result in image classification or natural language processing is reported without a held-out test set, and overfitting to the training set is treated as a basic methodological failure to be detected and corrected. Reinforcement learning, by contrast, spent its early years without an equivalent convention, in large part because the canonical benchmarks, such as Atari games, MuJoCo, continuous control tasks and single board-game environments, typically offer only one version of the environment to train and evaluate on.

Procedurally generated (PCG) environments close this gap. Instead of a single fixed level, the environment generator can be seeded to produce an effectively unlimited number of structurally similar but visually and spatially distinct instances. This makes the train/test split, standard practice in supervised learning for decades, directly applicable to RL: an agent is trained on a finite, fixed subset of generated levels and evaluated on levels drawn from the same generating distribution that it never encountered during training. Any gap between training and test performance is then direct, quantifiable evidence of memorization rather than learning.

This thesis uses Procgen (Cobbe et al., 2020), a suite of 16 procedurally generated game environments purpose-built for studying RL generalization, as its primary testbed. We train standard and modified RL algorithms on a restricted

set of training levels, evaluate them on the full, effectively unseen level distribution, and use the resulting generalization gap as the central dependent variable of the thesis. The guiding question, stated informally, is why RL agents fail when the environment changes only slightly, and which interventions, whether architectural, algorithmic or data-centric, actually close that gap.

1.2 Research Questions

The thesis is organized around three research questions. Each maps to a specific set of experimental conditions described in full in Chapter 3, and each is answered explicitly in Chapter 5.

RQ1: Algorithm family. How does the generalization gap differ between policy-based methods (PPO) and value-based methods (DQN) trained under matched conditions on the same games?

RQ2: Data augmentation. To what extent does adaptive, bandit-based data augmentation (UCB-DrAC) reduce the generalization gap relative to a vanilla PPO baseline?

RQ3: Curriculum learning versus augmentation. Does curriculum-based level selection (PLR) address a different failure mode from data augmentation, and do the two combine additively, synergistically or redundantly?

1.3 Contributions

Through this work, we have effectively created a systematic, statistically rigorous comparison of the four RL generalization approaches actually trained, namely a PPO baseline, a DQN baseline, UCB-DrAC and PLR + DrAC, across six of the sixteen Procgen games, using IQM and stratified bootstrap confidence intervals as recommended by Agarwal et al. (2021) rather than raw mean and standard deviation. This, in turn, provides us with a direct head-to-head comparison between the curriculum-learning family (PLR) and the data-augmentation family (UCB-DrAC) of generalization techniques, including their combination, characterizing what each does and does not address and allows us to extract a practical, evidence-based set of recommendations (Section 5.4) for practitioners training RL agents intended to generalize to unseen procedurally generated content.

Where the published literature already reports some of these comparisons individually, for example DrAC against PPO or PLR against PPO, this thesis's contribution is the consistent, controlled comparison of all of them under one

experimental pipeline, one set of hyperparameters and one statistical methodology, which removes the confound of differing codebases, hyperparameter budgets and evaluation protocols that makes cross-paper comparison in this literature unreliable.

1.4 Thesis Structure

Chapter 2 reviews the RL background needed to follow the rest of the thesis: the policy-gradient and value-based algorithm families, the generalization problem and its relationship to procedural content generation, the specific techniques studied, namely DrAC, UCB-DrAC and PLR, and the statistical methodology used to report results. Chapter 3 describes the experimental methodology in full, covering environment configuration, network architectures, the training pipeline, the evaluation protocol and the computational setup, in enough detail to reproduce the experiments. Chapter 4 presents the results without extensive interpretation: baseline performance and the generalization techniques compared head-to-head. Chapter 5 interprets these results, answers the three research questions directly and discusses the practical and theoretical implications. Chapter 6 summarizes the contributions, states the thesis's limitations plainly and outlines directions for future work.

Chapter 2: Background and Related Work

2.1 Reinforcement Learning Foundations

2.1.1 The RL Problem

Reinforcement learning formalizes sequential decision-making as a Markov Decision Process (MDP), defined by a tuple ***(S, A, P, R, γ):*** a state space ***S***, an action space ***A***, a transition function ***P(s' | s, a)*** giving the probability of reaching state ***s'*** from state ***s*** under action ***a***, a reward function ***R(s, a)***, and a discount factor ***γ*** ∈ [0, 1] that determines how much the agent values future reward relative to immediate ones. An agent's behaviour is defined by a policy ***π(a | s)***, a possibly stochastic mapping from states to actions. Using these policies, its objective is to maximize the expected discounted return, ***E [Σ⍰ γᵗ r⍰].*** The value function ***Vπ(s)*** gives the expected return from state ***s*** under policy ***π***, and the action-value function ***Qπ (s, a)*** gives the expected return from taking action ***a*** in state ***s*** and following ***π*** thereafter.

In our runs, the state observed by the agent is a 64×64 RGB image rendered by the Procgen environment, and the action space is a small discrete set of 15 game-specific button combinations, since all Procgen games share this common 15-dimensional discrete action space. Because the rendered image at a given timestep is treated as fully observing the underlying game state (Section 3.2.2), the problem is modelled as a standard MDP rather than a partially observed MDP (POMDP), consistent with the original Procgen benchmark design.

2.1.2 Policy Gradient Methods and PPO

Policy gradient methods optimize the policy directly via gradient ascent on the expected return. The REINFORCE estimator forms the basis of this family: it scales the log-probability gradient of each action taken by the return that followed it, increasing the probability of actions that led to high reward. REINFORCE is unbiased but has high variance, because it uses the full episode return as the learning signal for every action in the episode regardless of which action was actually responsible for the outcome.

Actor-critic methods reduce this variance by introducing a learned critic, ***V(s)***, that estimates the value function and is used to compute an advantage, ***A(s, a) = Q(s, a) − V(s)***, an estimate of how much better an action was than the policy's average behaviour in that state. Generalized Advantage Estimation (GAE) (Schulman et al., 2016) computes this advantage as an exponentially weighted average of n-step temporal-difference residuals, controlled by a parameter ***λ*** □ [0, 1]: a ***λ*** close to 1 weights long-horizon, low-bias but high-variance estimates more heavily, while a ***λ*** close to 0 weights the critic's own lower-variance, higher-bias estimate more heavily.

Proximal Policy Optimization (PPO) (Schulman et al., 2017) is the actor-critic algorithm used as the primary policy-based method in this thesis. PPO's central contribution is the clipped surrogate objective, which bounds how far a policy update can move the action-probability ratio ***r(θ) = πθ(a | s) / πθold(a | s)*** away from 1. Concretely, PPO maximizes ***min(r(θ)·A, clip(r(θ), 1−ε, 1+ε)·A)***, which removes the incentive to push the policy ratio outside the trust region ***[1−ε, 1+ε]*** once the sign of the advantage is already being exploited. This keeps policy updates conservative without the computational cost of the second-order trust-region methods (TRPO) that preceded it. The full PPO loss combines three terms: the clipped policy loss, a value-function regression loss, that is the mean squared error between predicted and observed returns, and a small entropy bonus that discourages premature convergence to a deterministic policy. PPO is the standard baseline for Procgen generalization research and was used in the original Procgen work and the subsequent DrAC and PLR papers, which makes it the natural baseline here as well (Cobbe et al., 2020; Raileanu et al., 2021; Jiang et al., 2021).

2.1.3 Value-Based Methods and DQN

Value-based methods, in contrast to policy-gradient methods, learn an estimate of the Q-function and derive a policy implicitly by acting greedily with respect to it. Deep Q-Networks (DQN) (Mnih et al., 2015) parameterize ***Q(s, a)*** with a convolutional neural network and train it by minimizing the temporal-difference error against a bootstrapped target, ***y = r + γ · max$_{a'}$ Q(s', a')***.

Two stabilization mechanisms distinguish DQN from naive Q-learning with function approximation. The replay buffer stores past transitions ***(s, a, r, s')*** and trains on randomly sampled minibatches from this buffer rather than on the most recent transition, breaking the temporal correlation between consecutive samples and allowing each transition to be reused for multiple gradient updates. The target network is a periodically updated, frozen copy of the Q-network used to compute the bootstrap target ***y***, which prevents the network from chasing a target that moves with every gradient step. Exploration is handled separately from the value estimate via an ***ε***-greedy policy: with probability ***ε*** the agent acts uniformly at random, and with probability ***1 − ε*** it acts greedily, and ***ε*** is annealed from a high initial value toward a small floor over the course of training.

DQN is included in this thesis specifically because its replay buffer is expected to interact badly with procedural generalization: by training repeatedly on the same stored transitions from a fixed pool of training levels, DQN has more opportunity to memorize level-specific detail than PPO, which by default uses each rollout once before discarding it. This thesis treats the PPO/DQN comparison (RQ1) as a direct test of whether sample-reuse strategy, independent of any explicit generalization technique, is itself a generalization-relevant design choice.

2.2 Generalization in Reinforcement Learning

2.2.1 The Overfitting Problem in RL

Overfitting in supervised learning is well understood: a model with enough capacity can fit the training set's idiosyncrasies, such as label noise or spurious correlations, rather than the underlying signal, and this is detected by a gap between training and held-out performance. RL overfitting takes a related but mechanistically distinct form. An RL agent can achieve a high training return not by learning a transferable strategy but by memorizing a sequence or policy that is locally optimal for the specific level layouts, enemy placements and visual textures encountered during training, which is closer to memorizing the answers to a fixed quiz than to learning the underlying subject.

This failure mode is compounded by two properties specific to RL. First, the agent itself determines the distribution of states it observes during training, since its own policy induces the data distribution, so as it specializes to the training levels it

may stop exploring the states that would reveal the failure of its specialized strategy. Second, because reward is typically sparse and delayed relative to the action that earned it, an agent can latch onto superficial correlations between visual features and reward, for example a rule such as jump whenever the background is dark in one specific Ninja level, that have no causal relationship to the actual game mechanics. Such correlations are invisible from the training curve alone and reveal themselves only as a drop in performance once the same visual feature appears in a different structural context.

The empirical study of this failure mode now has a substantial literature, recently consolidated by Kirk et al. (2023) into a taxonomy of zero-shot generalisation problems and methods. Early evidence that RL agents memorise rather than generalise came from Zhang et al. (2018), who showed that agents trained on a fixed set of procedurally generated maze layouts could achieve near-perfect training performance while failing badly on held-out layouts, and from Packer et al. (2018), who proposed systematic train/test protocols across several continuous-control and discrete tasks. Cobbe et al. (2019) made the same point quantitatively on the purpose-built CoinRun environment, demonstrating that agents continue to overfit even when trained on hundreds or thousands of levels, and that the size of the training set needed to close the gap is far larger than intuition would suggest. These studies collectively reframed generalisation from an afterthought into a primary axis of evaluation and motivate the train/test protocol adopted in this thesis.

2.2.2 Procedural Content Generation as a Solution

Procedural content generation (PCG) directly addresses both properties above. By generating each level from a random seed under a fixed generative grammar, PCG produces a well-defined, effectively infinite distribution of structurally related but superficially distinct environment instances. Training on a finite, fixed subset of seeds (e.g. seeds 0–199) and evaluating on disjoint seeds drawn from the same distribution gives a train/test split with the same logical structure as the supervised-learning case: any drop in performance on the held-out seeds is, by construction, attributable to a failure to generalize beyond the specific training instances, rather than to environment non-stationarity or evaluation noise.

2.2.3 The Procgen Benchmark

Procgen (Cobbe et al., 2020) is a suite of 16 procedurally generated arcade-style game environments developed by OpenAI specifically to study generalization in deep RL. Every game shares a common interface: a 64×64 RGB pixel observation, a small discrete action space, and a procedural level generator controlled by an integer seed. Procgen environments are implemented in C++ for speed and support fast vectorized stepping of dozens to hundreds of parallel environment instances, which is necessary given the very large number of training steps (tens of millions) typically required for RL convergence.

Procgen defines two distribution modes: easy and hard, which differ in the difficulty and visual/structural diversity of generated levels. This thesis uses easy mode throughout (Section 3.2.1), consistent with the majority of published generalization studies on this benchmark, and with hard mode noted as a direction for future work (Section 6.3). The standard Procgen evaluation protocol trains on a restricted number of levels (a common convention is 200 or 500) and evaluates on the full level distribution (in principle, the full space of seeds), which this thesis follows.

The 16 games span a deliberately broad range of game mechanics, from platforming in environments such as CoinRun, Ninja, Climber and Jumper, through shooting and reflex control in StarPilot and BossFight, to navigation and pathfinding in Maze, Heist and CaveFlyer and resource collection or avoidance in BigFish, Miner, FruitBot, Leaper, Chaser, Dodgeball and Plunder, so that no single technique can be judged on the basis of one game's idiosyncrasies. The full per-game descriptions and the rationale for including each are given in Appendix A.1.

Beyond providing the benchmark, Cobbe et al. (2019, 2020) also established several of the empirical regularities this thesis builds on: that deeper, residual IMPALA-style encoders generalise better than the shallow Nature-CNN of the original DQN, and that regularisers imported from supervised learning, such as L2 weight decay, dropout, batch normalisation and, most relevant here, data augmentation, measurably narrow the generalisation gap. Farebrother et al. (2018) reported the analogous result for value-based methods, showing that

standard supervised-learning regularisers improve the transfer of DQN agents across Atari game modes. The methods compared in Chapter 4 can be read as more sophisticated, RL-specific descendants of these simple regularisers.

2.2.4 Other Benchmarks

MiniGrid (Chevalier-Boisvert et al., 2023), a Farama Foundation suite of simple procedurally generated grid-world tasks, is mentioned here as a secondary benchmark relevant to interpreting Procgen results, though it is not part of this thesis's primary experimental scope (Section 6.2). MiniGrid's key methodological value is that it can be run with either symbolic observations (the agent sees abstract object tokens directly, with no pixels to overfit to) or rendered pixel observations. This separates visual memorization from structural memorization: if an agent fails to generalize even under symbolic observations, the failure cannot be attributed to overfitting on textures or color schemes, and must instead reflect a deeper issue with how the agent represents the task. GVGAI is mentioned for completeness as a further alternative benchmark with a similar procedural philosophy but is likewise outside this thesis's experimental scope.

2.3 Methods for Improving Generalization

2.3.1 Data Augmentation in RL

Data augmentation imports a standard computer-vision regularization technique into the RL setting: the agent is trained on randomly transformed versions of the same observations, for instance cropped, colour-jittered or flipped, so that the network cannot rely on transformation-sensitive features that should be irrelevant to the optimal policy.

RAD (Reinforcement learning with Augmented Data) (Laskin et al., 2020) is the simplest member of this family: it applies a random augmentation to observations before they are passed to the network during the PPO update, with no change to the loss function itself. The augmentations evaluated in this thesis (Section 3.4.3) follow the original RAD and DrAC papers: random crop (pad then crop at a random offset, encouraging robustness to small viewport shifts), color jitter (random brightness/contrast/saturation/hue perturbation, discouraging reliance on fixed color schemes), random grayscale, random cutout (randomly occlude a rectangular patch), horizontal flip, and random convolution (apply a randomly

initialized convolutional filter, which aggressively perturbs visual style while preserving spatial structure).

DrAC (Data-regularized Actor-Critic) (Raileanu et al., 2021) extends RAD with two explicit consistency regularization terms added to the PPO loss: a policy (actor) consistency term, the KL divergence between the action distribution computed on the original observation and on the augmented observation, and a value (critic) consistency term, the mean squared error between the value estimates on original and augmented observations. Where RAD merely hopes the network learns invariance from augmented training data alone, DrAC explicitly penalizes any inconsistency between the policy's (and critic's) behavior on the two views, which the original paper shows yields a stronger invariance and a larger improvement over PPO.

UCB-DrAC (Raileanu et al., 2021) addresses a practical limitation of DrAC and RAD: the best-performing augmentation is game-dependent, and no single augmentation dominates across all 16 Procgen games. UCB-DrAC frames augmentation selection as a multi-armed bandit problem, where each of the candidate augmentations is an arm. At each PPO update it selects an augmentation using the UCB1 rule, ***Score = mean reward + c·√(ln(total pulls) / pulls for this arm)***, balancing exploitation of the empirically best-performing augmentation against exploration of less-tried ones, and updates the bandit's statistics using the resulting training performance. This automates the augmentation choice that would otherwise require a manual per-game hyperparameter search, and is the augmentation method this thesis treats as its primary point of comparison against PLR (RQ2, RQ3).

Data augmentation in RL is closely related to two neighbouring ideas. The first is domain randomisation (Tobin et al., 2017), which deliberately randomises the appearance and dynamics of a simulator so that a policy trained in simulation transfers to the real world; augmentation can be seen as a lightweight, in-distribution form of the same principle. The second is network randomisation (Lee et al., 2020), which perturbs visual style by passing observations through a randomly initialised convolutional layer, the mechanism behind the random_conv augmentation used here. The key conceptual distinction this thesis draws on is

between RAD, which augments the observations fed to the policy and merely hopes invariance emerges, and DrAC, which leaves the policy-gradient objective on the original observations and adds an explicit consistency penalty. The latter avoids the biased gradient that input augmentation can introduce (Raileanu et al., 2021), which is why DrAC and its automatic variant UCB-DrAC, rather than RAD, are the augmentation methods evaluated in this thesis.

2.3.2 Curriculum Learning: Prioritized Level Replay

Prioritized Level Replay (PLR) (Jiang et al., 2021) takes an orthogonal approach to the augmentation methods above: rather than changing what the agent sees, it changes which levels the agent is trained on. Instead of sampling training levels uniformly, PLR maintains a priority score per level based on the L1 value loss, that is the absolute difference between the critic's predicted return and the actually observed return for that level, as a proxy for how much the agent still has to learn from it.

Level sampling combines two sources with probability ρ (typically 0.5): with probability ρ, a level is drawn from the priority buffer using rank-based sampling, where higher-priority levels are more likely to be selected and a temperature parameter β controls how sharply this preference is applied; with probability $1-\rho$, a level is drawn uniformly from the full training set, which prevents the curriculum from collapsing onto a small subset of consistently hard levels. A staleness bonus additionally upweights levels that have not been sampled recently, ensuring that levels which have stopped being prioritized are still periodically revisited rather than forgotten entirely.

PLR's contrast with the augmentation methods is conceptually important to this thesis: where DrAC and UCB-DrAC change the representation of a given experience (forcing the network to be invariant to superficial visual perturbation), PLR changes the distribution of experiences the agent is trained on (ensuring practice time is spent on levels that are actually informative). Because these are different mechanisms targeting different failure modes, the literature has found they can be combined, motivating the PLR + UCB-DrAC condition central to RQ3.

PLR also sits within the broader programme of automatic curriculum learning and unsupervised environment design (UED). Where PLR re-weights a fixed pool of

levels, methods such as PAIRED (Dennis et al., 2020) and ACCEL (Parker-Holder et al., 2022) go further and adapt the level generator itself, using a regret signal to propose levels at the frontier of the agent's competence. These methods are more powerful but also substantially more complex and compute-intensive; PLR's appeal for this thesis is that it improves the training distribution using only a value-based scoring rule over the levels the environment already provides, making it a fair, low-overhead point of comparison against augmentation. Its adversarial and generative extensions are returned to as future work (Section 6.3).

2.3.3 Other Approaches

For completeness, several other generalisation techniques from the literature are situated here relative to the augmentation and curriculum families, although they fall outside this thesis's experimental scope (Section 6.2). Network randomisation and the associated random-feature-matching objective (Rand-FM; Lee et al., 2020) randomise early convolutional layers per episode and encourage the agent to match features across the randomised and clean views, directly targeting visual overfitting. IBAC-SNI (Igl et al., 2019) attacks the problem from an information-theoretic angle, combining an information bottleneck with selective noise injection so that the policy discards level-specific detail while remaining stable to train. Mixreg (Wang et al., 2020) adapts the mixup regulariser to RL, training on convex interpolations of observations and their associated supervision signals to impose a smoother, more linear policy over the observation space.

These approaches are largely complementary to the two families studied here: they mostly change the representation an agent learns (like augmentation) or the noise it is trained under, rather than the distribution of experiences it sees (like curriculum learning). They are noted to situate UCB-DrAC and PLR within the wider landscape rather than to present those two as the only available options; a controlled comparison against them is left as future work.

2.4 Statistical Evaluation in Deep RL

Agarwal et al. (2021), “Deep Reinforcement Learning at the Edge of the Statistical Precipice”, argues that the field's standard practice of reporting mean ± standard deviation across a handful of random seeds is statistically unreliable, given the high run-to-run variance characteristic of deep RL and the small number of seeds

(often 3-5) typically used due to computational cost. Two specific problems motivate this thesis's evaluation methodology (detailed fully in Section 3.5).

First, aggregating performance across games of very different score scales, for example StarPilot scores in the tens to hundreds against CoinRun scores near 10, using a simple mean lets high-scale games dominate the aggregate and obscures real improvements or regressions on lower-scale games. The Interquartile Mean (IQM), the mean of the middle 50% of results after discarding the top and bottom quartiles, is more robust to this kind of scale imbalance and to individual outlier runs, once scores are first normalized per game to a [0,1] scale. Second, with only a handful of seeds, point estimates of the mean or IQM carry substantial sampling uncertainty that a single number obscures; stratified bootstrap confidence intervals, which resample within each game before aggregating and repeat this tens of thousands of times, give an explicit uncertainty range, allowing a claim such as an IQM of 0.62 with a 95% confidence interval of [0.55, 0.69] rather than an unqualified point estimate.

The motivation for this methodology is not merely statistical hygiene. Henderson et al. (2018) showed that headline deep-RL results can reverse under different random seeds, hyperparameters, or even codebases, and that the field's then-standard practice of reporting the mean of a handful of runs systematically overstated the reliability of reported improvements. Agarwal et al. (2021) quantified the same problem and proposed the interquartile mean with stratified bootstrap confidence intervals as a more robust alternative. With only three seeds per condition, this thesis is squarely in the regime those papers warn about, which is why every aggregate comparison in Chapter 4 is reported with an explicit confidence interval and why overlapping intervals are treated as inconclusive rather than as evidence of a real difference.

This thesis adopts both recommendations via the rliable library (Agarwal et al., 2021), reporting IQM with bootstrap confidence intervals, performance profiles (the fraction of game/seed combinations exceeding a given normalized score threshold, across the full threshold range), and probability-of-improvement estimates (the probability that method A outperforms method B on a randomly drawn game/seed pair) for all head-to-head comparisons in Chapter 4. This

choice is made explicitly here, before the reader encounters these statistics in Chapter 4, so that the aggregate figures in Section 4.2.3 can be read without re-deriving their justification.

Chapter 3: Methodology

3.1 Experimental Design

3.1.1 Experimental Scope

Given the computational cost of training reinforcement-learning agents for 25M timesteps per run (Section 3.6), the experiments were scoped to the core set of method comparisons needed to answer RQ1–RQ3 under matched conditions. A broader set of ablation studies was designed but, owing to a fixed time budget, was not executed, and is described as future work in Section 6.3. Every run reported here uses the same training pipeline, hyperparameters and evaluation protocol.

The experiments comprise 66 completed, deduplicated runs that directly answer RQ1–RQ3 at the level of the core method comparison:

- PPO IMPALA-CNN baseline: 6 games × 3 seeds = 18 runs
- PPO + UCB-DrAC: 6 games × 3 seeds = 18 runs
- PPO + PLR + DrAC: 6 games × 3 seeds = 18 runs
- DQN baseline: 4 games × 3 seeds = 12 runs

All reported numbers come from this set. The six games studied are BigFish, CoinRun, FruitBot, Miner, Ninja and StarPilot, while the DQN baseline uses the four-game representative subset of BigFish, CoinRun, Ninja and StarPilot.

3.1.2 Hyperparameter Choices

Table 3.1 lists the PPO hyperparameters used for every PPO-family run, namely the PPO baseline, UCB-DrAC and PLR + DrAC, taken from ppo_procgen_baseline.yaml. These values follow the CleanRL Procgen baseline configuration (Huang et al., 2022) and the original Procgen paper's recommended settings, and are held fixed across all games and all techniques: the explicit goal of this thesis is to compare methods under matched conditions, not to tune hyperparameters per game, which would confound the comparison.

Table 3.1 - *PPO hyperparameters, fixed across all games and PPO-family conditions*

Hyperparameter	Value	Justification
Total timesteps	25,000,000	Standard Procgen training budget (Cobbe et al., 2020)
Parallel environments	64	Matches EnvPool throughput sweet spot (Section 3.2.1)
Rollout length (num_steps)	256	CleanRL/Procgen baseline default
PPO epochs per update	3	CleanRL/Procgen baseline default
Minibatches per epoch	8	CleanRL/Procgen baseline default
Learning rate	5e-4	CleanRL/Procgen baseline default
Discount factor (γ)	0.999	Long-horizon Procgen episodes; matched to reward normalization (Section 3.2.3)
GAE λ	0.95	CleanRL/Procgen baseline default
Clip coefficient (ε)	0.2	Standard PPO default
Entropy coefficient	0.01	CleanRL/Procgen baseline default
Value loss coefficient	0.5	CleanRL/Procgen baseline default
Max gradient norm	0.5	Standard PPO default
Training levels (num_levels)	200	Standard Procgen generalization protocol
Distribution mode	easy	Section 2.2.3

Table 3.2 - *DQN hyperparameters, fixed across all DQN baseline runs*

Hyperparameter	Value	Justification
Total timesteps	25,000,000	Matched to the PPO budget (25M)
Replay buffer size	200,000	1M ideal; capped by the 26 GB WSL2 RAM allocation
Target network update interval	10,000	Hard target-network update interval (steps)
ε schedule (start → end, decay steps)	1.0 → 0.02 over 1,000,000 steps (linear)	Standard ε-greedy exploration schedule
Learning rate	1.0e-4	Lower than PPO; more stable with a replay buffer
Batch size	32	Standard DQN minibatch size
Discount factor (γ)	0.99	Lower than PPO's 0.999 — very high γ is less stable with a replay buffer

3.2 Environment Setup

3.2.1 Procgen Configuration

All experiments use Procgen (Section 2.2.3) in easy distribution mode, with 200 levels, namely seeds 0 to 199, reserved for training and the full level distribution used for test-time evaluation. The PPO-family methods are evaluated on six representative games, namely BigFish, CoinRun, FruitBot, Miner, Ninja and StarPilot, and the DQN baseline on a four-game subset of BigFish, CoinRun, Ninja and StarPilot. Environment stepping is handled by EnvPool (Weng et al., 2022), a C++ vectorized environment backend chosen for its substantially higher throughput than the native Procgen Python bindings; internal benchmarking measured roughly 20,000 steps per second for raw EnvPool stepping against 5,000 to 10,000 for native Procgen. Each training run uses 64 parallel environment instances, balancing rollout-collection throughput against per-environment memory overhead on the available hardware (Section 3.6.1).

One environment-integration detail is worth recording for reproducibility: EnvPool does not register Procgen tasks under the procgen:procgen-<game>-v0 identifier used by the native Procgen package. Instead, the distribution mode is baked directly into the EnvPool task identifier, in the form {GameCapitalized}{ModeCapitalized}-v0 (for example, CoinrunEasy-v0 or StarpilotHard-v0), and is not passed as a separate distribution_mode keyword argument to envpool.make(). The pipeline's environment-backend detection logic checks for this task-ID pattern explicitly rather than searching for a “procgen” substring.

3.2.2 Observation Preprocessing

Observations are 64×64 RGB images, delivered by the environment as uint8 arrays and normalised to [0,1] inside the network's forward pass rather than in the environment wrapper, so that the data transferred between CPU and GPU stays in compact uint8 buffers for as long as possible (see also the GPU-transfer optimisations in Section 3.4.1). No frame stacking is used: Procgen environments are designed to be fully observable from a single frame, so the information needed to act optimally is present in one rendered image and stacking previous frames would add input dimensionality without adding usable information. This keeps the observation tensor small, which matters because rollout buffers of tens of thousands of frames must be held in GPU memory during each PPO update.

3.2.3 Reward Normalization

Rewards are normalised using a running estimate of the standard deviation of the discounted return, following the standard CleanRL-style PPO implementation. The normaliser's implicit discount factor is set to exactly match PPO's own $\gamma = 0.999$. This match matters: if the normaliser's effective discount differed from the discount used in the agent's GAE computation, the scale of the advantage estimates the agent learns from would no longer match the scale of returns the value function is trained to predict, which can destabilise training in ways that are difficult to diagnose from the training curves alone. In this project the two discount values were kept identical by construction, so this failure mode did not arise. Evaluation environments, by contrast, apply no reward normalisation, so the returns reported throughout Chapter 4 are in the game's native units.

3.3 Neural Network Architectures

3.3.1 IMPALA-CNN (primary encoder)

The primary visual encoder used across all experiments is the IMPALA-CNN architecture (Espeholt et al., 2018), consisting of three residual blocks with channel widths 16, 32 and 32, each block comprising a convolution followed by max-pooling and two residual sub-blocks, then a ReLU nonlinearity, an adaptive average pool and a final fully connected layer projecting to a 256-dimensional representation. This encoder has 626,256 trainable parameters in total. That figure is for the full actor-critic with a 15-action head, since the IMPALA encoder on its own has 622,144 parameters and the policy and value heads add a further 4,112. The encoder is shared by the actor and critic heads in the ActorCritic module, with separate small linear heads producing the action-distribution logits and the scalar value estimate respectively.

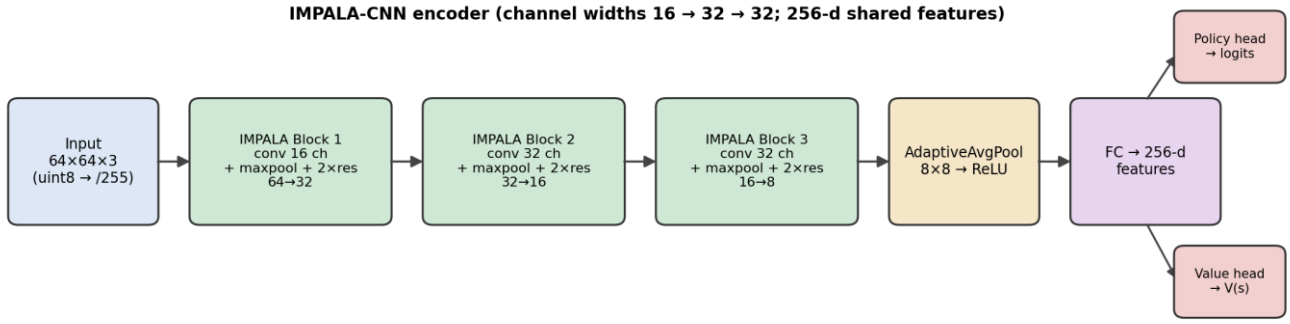

Figure 3.1 - *IMPALA-CNN encoder: three convolution + max-pool blocks (channel widths 16 → 32 → 32), each followed by two residual blocks, then an adaptive 8×8 average pool and a fully connected projection to a 256-dimensional feature vector shared by the policy and*

3.4 Training Pipeline

3.4.1 PPO Training Loop

Each PPO update cycle collects a rollout of num_steps = 256 timesteps across all 64 parallel environments, giving 16,384 environment-steps per rollout, computes advantages via GAE, and performs num_epochs = 3 epochs of minibatch gradient descent over num_minibatches = 8 minibatches per epoch.

3.4.2 DQN Training Loop

The DQN training loop follows the standard structure described in Section 2.1.3: actions are selected $\boldsymbol{\varepsilon}$-greedily with respect to the online Q-network, transitions are stored in a replay buffer, and minibatches are sampled uniformly at random

for each gradient step against a periodically synchronized target network. The replay buffer is an in-memory, uniformly sampled buffer of 200,000 transitions (no prioritized replay).

3.4.3 Augmentation Integration (DrAC and UCB-DrAC)

Augmentation is applied during the PPO update phase, not during environment interaction: the rollout is collected using un-augmented observations (so the agent's actual behaviour in the environment, and the resulting reward signal, are never distorted by the augmentation), and the augmentation is applied only to the minibatches used to compute the policy and value losses. DrAC passes both the original and an augmented copy of each minibatch through the network and compares the resulting policy and value outputs via the KL-divergence and MSE consistency terms described in Section 2.3.1, added to the standard PPO loss with a single weighting coefficient aug_coef = 0.1 applied equally to both terms (drac_mode = full). UCB-DrAC wraps this same DrAC update with a per-update bandit step that chooses which single augmentation to apply that update, using the UCB1 score described in Section 2.3.1, with exploration coefficient c = 0.1. (RAD, the simpler precursor that augments the network's inputs without an explicit consistency penalty, is described in Section 2.3.1 but was not run as a separate condition.)

3.4.4 PLR Integration

PLR overrides the default uniform level sampling at the point where each parallel environment instance calls env.reset(): instead of sampling the next level uniformly from the training set, the level selection logic (Section 2.3.2) is consulted, drawing from the priority buffer with probability $\boldsymbol{\rho}$ and uniformly otherwise. After each level is played, its priority score is updated using the L1 value loss, that is the absolute difference between the critic's predicted return for that level, recorded when it was last played, and the actual return achieved, with staleness bonuses applied as described in Section 2.3.2. The as-run values are $\boldsymbol{\rho}$ = 0.5 and $\boldsymbol{\beta}$ = 0.1, with a staleness coefficient of 0.1 and L1-value-loss scoring, taken from ppo_procgen_plr_drac.yaml.

3.5 Evaluation Protocol

3.5.1 Train-Level vs. Test-Level Evaluation

Evaluation is performed using separate evaluation environment instances, distinct from the training environments, configured without reward normalization (so that reported returns are in the environment's native, interpretable scale rather than the agent's internally normalized training scale). The evaluation policy is deterministic (argmax over the action distribution rather than sampling), removing policy stochasticity as a source of evaluation variance so that any remaining variance reflects genuine differences between levels or seeds rather than sampling noise in action selection. Each evaluation point (recorded every eval_freq = 500,000 timesteps) runs 100 episodes; train-level evaluation draws levels from the same 200-level training set used during training, and test-level evaluation draws from the full, effectively unseen level distribution.

3.5.2 Metrics

Three primary per-game metrics are reported, defined formally in Section 2.4 and Chapter 4: episodic return; the generalization gap (mean train return minus mean test return); and overfitting curves (train and test return as a function of training progress). For the aggregate comparison (Section 4.2.3), per-game scores are additionally normalized to a [0,1] scale, where 0 corresponds to a random policy's performance and 1 to the best performance achieved by any method under test on that game. The per-game random-policy scores are the fixed RANDOM_BASELINES constants defined in analysis/rliable_analysis.py rather than re-measured per run.

3.5.3 Statistical Methodology

All executed conditions use 3 random seeds per condition (seeds 1–3); the ablation studies were not run (Section 3.1.2). All aggregate comparisons across games and seeds use the rliable methodology described in Section 2.4: IQM with stratified bootstrap 95% confidence intervals (50,000 bootstrap resamples), performance profiles, and probability-of-improvement estimates with bootstrap confidence intervals for pairwise method comparisons.

Chapter 4: Results

All figures and tables in this chapter report the complete set of results: the PPO baseline, PPO + UCB-DrAC and PPO + PLR + DrAC on the six Procgen games BigFish, CoinRun, FruitBot, Miner, Ninja and StarPilot, and the DQN baseline on the four-game representative subset of BigFish, CoinRun, Ninja and StarPilot, each averaged over three seeds. Following Agarwal et al. (2021), aggregate comparisons use the interquartile mean (IQM) of per-game normalized scores with 95% stratified-bootstrap confidence intervals (Section 4.2.3).

4.1 Baseline Performance

4.1.1 PPO IMPALA-CNN Baseline

Table 4.1 reports per-game train and test return for the PPO IMPALA-CNN baseline, averaged over three seeds, for the six games studied.

Table 4.1 *- PPO IMPALA-CNN baseline: per-game train and test return and generalization gap (mean ± standard deviation across 3 seeds).*

Game	Train Return	Test Return	Gen. Gap
BigFish	4.50 ± 1.72	1.59 ± 0.49	2.91 ± 1.24
CoinRun	9.73 ± 0.06	8.67 ± 0.15	1.07 ± 0.21
FruitBot	29.34 ± 0.62	23.86 ± 0.63	5.48 ± 0.66
Miner	12.83 ± 0.07	9.71 ± 0.48	3.12 ± 0.45
Ninja	8.10 ± 0.36	6.03 ± 0.50	2.07 ± 0.40
StarPilot	29.68 ± 2.26	27.90 ± 2.93	1.79 ± 0.89

In the same table, we also report the mean and standard deviation across the three seeds. The baseline attains its highest absolute test returns on FruitBot (23.86) and StarPilot (27.90) and its lowest on BigFish (1.59); the latter falls below the range reported by Cobbe et al. (2020) and is consistent with BigFish's difficulty as a size-discrimination task (Section 2.2.3). Every game shows a positive generalization gap, confirming that even a well-tuned PPO baseline overfits to its 200 training levels.

Figures below trace train return and test return along with the resulting generalisation gap over the 25M-timestep run, for four representative games and all available methods. The persistent vertical separation between the train and test curves is the generalisation gap made visible: where the two curves diverge and stay apart (most clearly on BigFish) the agent is overfitting to its 200 training levels, whereas near-coincident curves (CoinRun) indicate a game that generalises easily. The panels also show that augmentation mainly raises the whole BigFish curve, while curriculum-style PLR + DrAC tends to flatten both

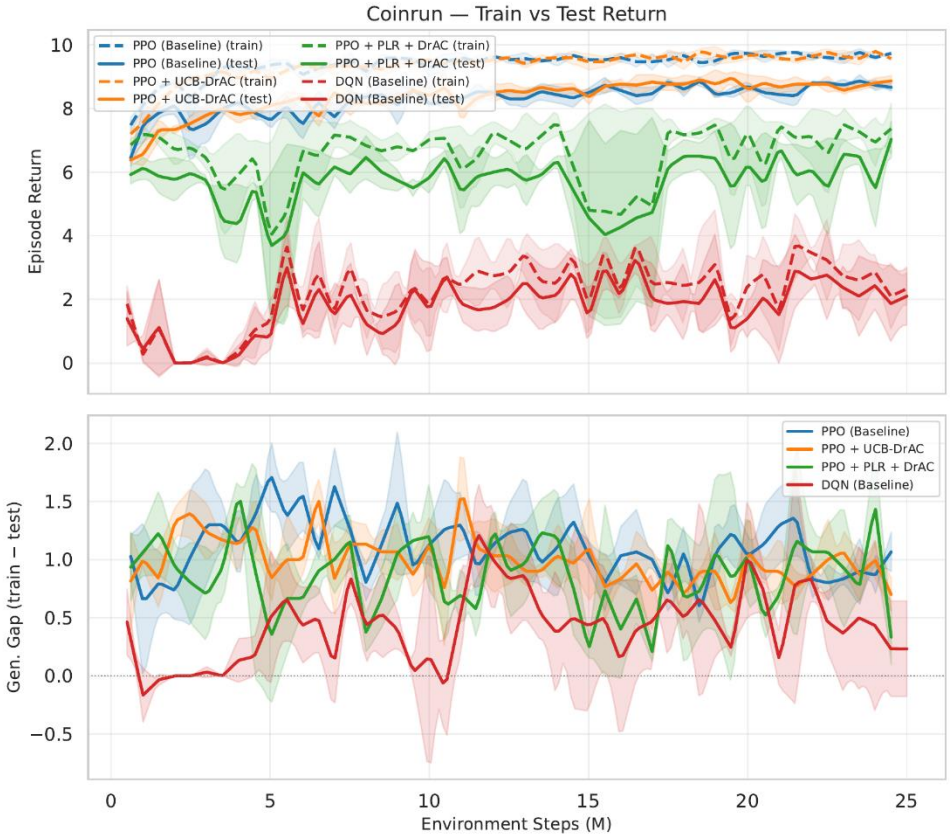

Figure 4.1 *- CoinRun: train and test episodic return (top) and generalization gap (bottom) vs. environment steps, all available methods, n=3 seeds for PPO/UCB-DrAC/PLR+DrAC/DQN.*

curves together.

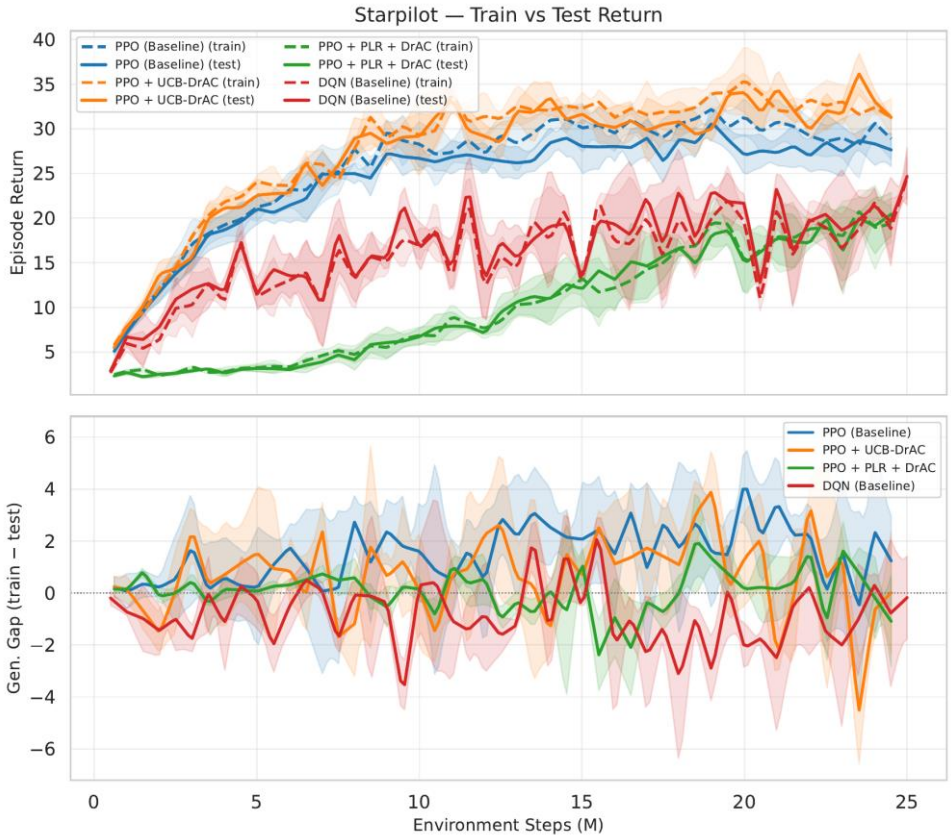

Figure 4.2 - *StarPilot: train and test episodic return (top) and generalization gap (bottom) vs. environment steps.*

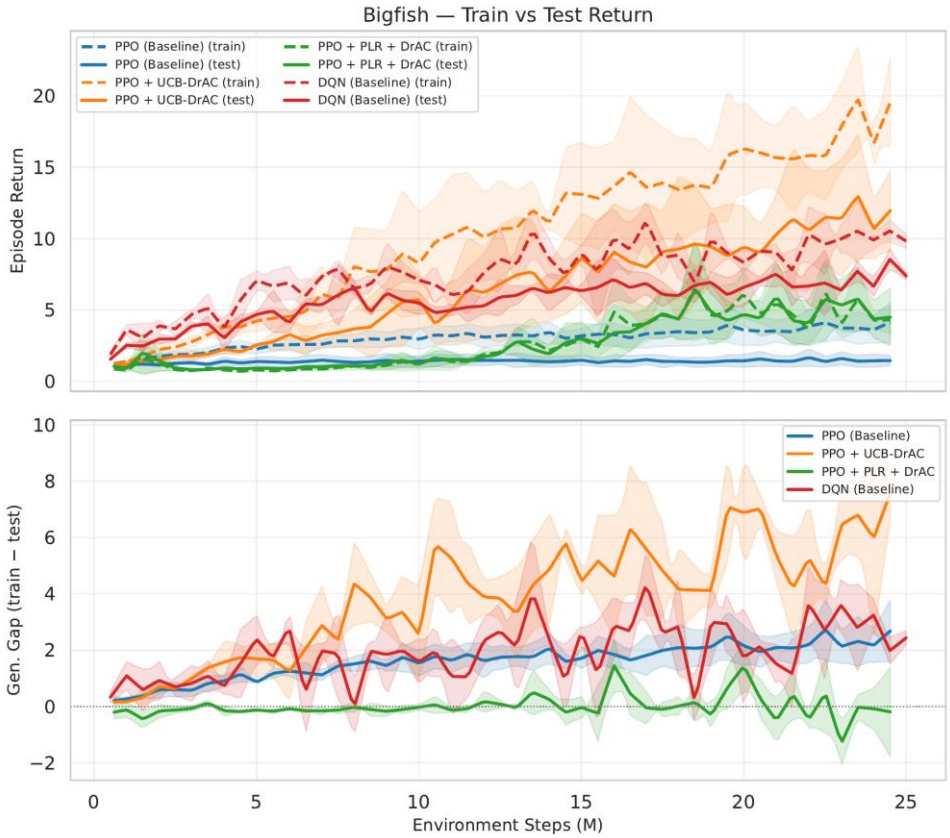

Figure 4.3 - *BigFish: train and test episodic return (top) and generalization gap (bottom) vs. environment steps.*

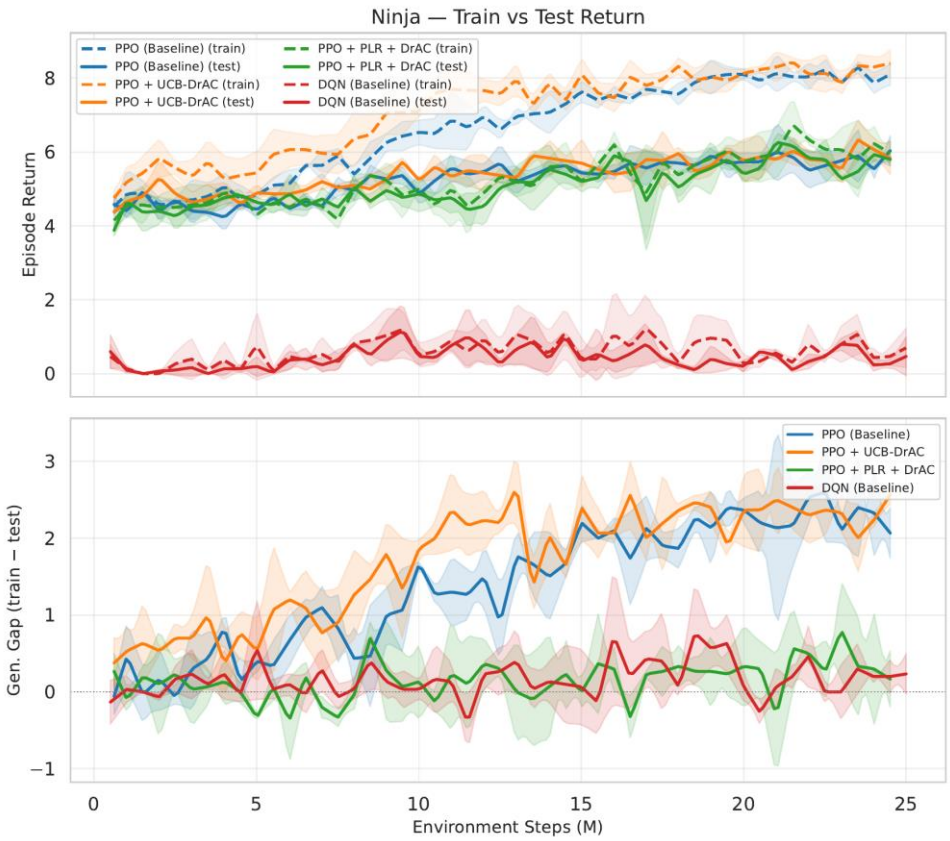

Figure 4.4 - *Ninja: train and test episodic return (top) and generalization gap (bottom) vs. environment steps.*

From the figures shown, we observe that PPO achieves substantial training return on all 6 completed games, but test performance show a persistent train–test gap on every game. That being said, the per-seed training-curve time series needed to pinpoint the exact plateau onset were not exported in this snapshot. FruitBot and StarPilot show the highest absolute test returns (23.9 and 27.6 respectively), with BigFish on the other hand showing the lowest (1.5), consistent with BigFish's relative difficulty as a size-discrimination task (Section 2.2.3).

4.1.2 PPO vs. DQN

Error! Reference source not found. compares PPO and DQN baselines on the 4 games for which DQN results are currently available: BigFish, CoinRun, Ninja, and StarPilot.

Table 4.2 - *PPO vs. DQN baseline: test return by game (mean across seeds; PPO n=3, DQN n=3).*

Game	PPO Test Return	DQN Test Return	Gap (PPO − DQN)
BigFish	1.5	7.4	−5.9
CoinRun	8.7	2.1	+6.6
Ninja	6.0	0.5	+5.5
StarPilot	27.6	24.7	+2.9

The results in said table show that PPO outperforms DQN on test return for three of the four shared games, namely CoinRun, Ninja and StarPilot, while DQN outperforms PPO on BigFish. This is a more mixed picture than the directional hypothesis in Section 2.1.3, that DQN's replay buffer should generally hurt generalization relative to PPO, would predict on its own, and it is examined further as part of RQ1 in Section 5.1. The per-game gaps confirm this: DQN's generalization gap is actually smaller than PPO's on all four shared games, at 2.44 against 2.91 on BigFish, 0.23 against 1.07 on CoinRun, 0.23 against 2.07 on Ninja and −0.18 against 1.79 on StarPilot. This mostly reflects DQN's much lower absolute returns, since it overfits less because it learns less, rather than

better generalization; the exception is BigFish, where DQN has both a higher test return and a smaller gap than PPO.

4.2 Generalization Techniques

Figure 4.5 gives an overview of final test return for every method × game combination with completed runs, before the per-technique breakdowns in Sections 4.2.1–4.2.2.

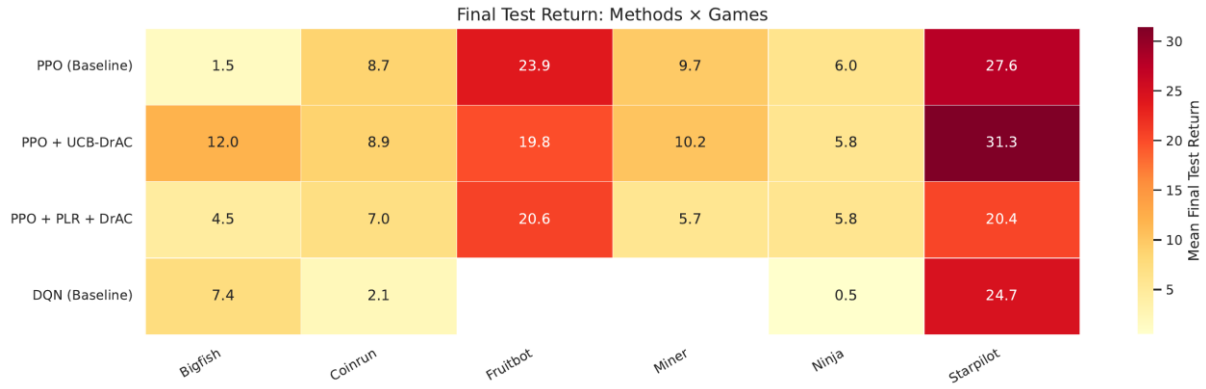

Figure 4.5 - *Final test return for every method × game combination (mean over three seeds). Blank cells indicate the two games outside the DQN subset (FruitBot and Miner), which DQN was not run on by design.*

Reading across each row of the heatmap shows how a single method performs over the six games; reading down each column shows which method wins on a given game. UCB-DrAC attains the best or joint-best test return on four of the six games, with its largest advantage on BigFish (12.0 versus the baseline's 1.5); PPO retains the lead on FruitBot and Ninja, and DQN is competitive only on StarPilot and BigFish, the two games where it learns appreciably at all. The blank FruitBot and Miner cells for DQN reflect the four-game DQN subset, not missing runs.

4.2.1 UCB-DrAC Results

Table 4.3 - *PPO baseline vs. PPO + UCB-DrAC: test return by game.*

Game	PPO (Baseline)	PPO + UCB-DrAC	Δ (UCB-DrAC − Baseline)
BigFish	1.5	12.0	+10.5
CoinRun	8.7	8.9	+0.2
FruitBot	23.9	19.8	−4.1
Miner	9.7	10.2	+0.5

Game	PPO (Baseline)	PPO + UCB-DrAC	Δ (UCB-DrAC − Baseline)
Ninja	6.0	5.8	−0.2
StarPilot	27.6	31.3	+3.7

Figure 4.6 shows the generalization gap (train minus test return) by game, grouped by method, for all four conditions with completed runs.

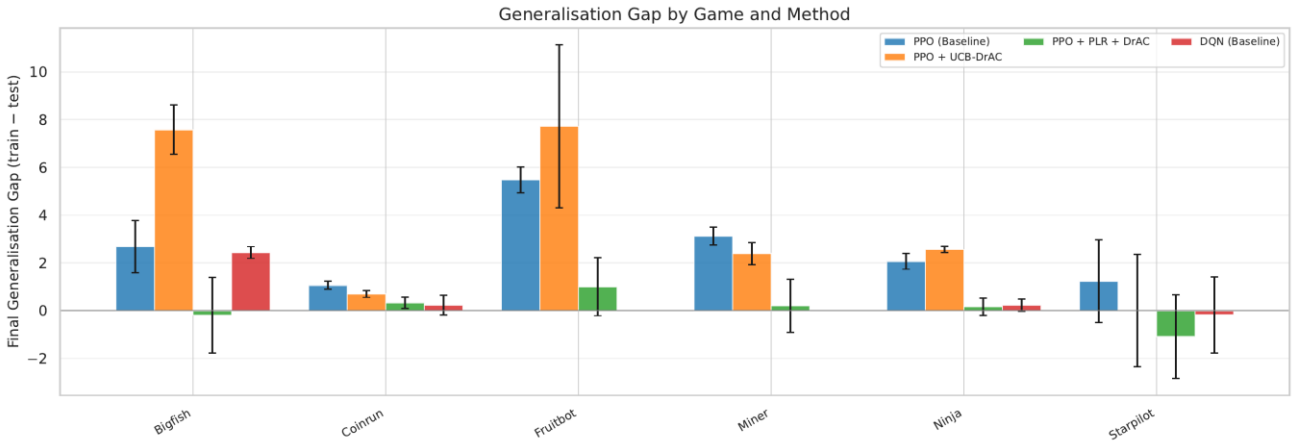

Figure 4.6 - *Final generalization gap by game and method, for the PPO baseline, PPO + UCB-DrAC, PPO + PLR + DrAC and the DQN baseline. Error bars show standard deviation across seeds.*

From the above, we observer that UCB-DrAC produces the largest single improvement of any technique on BigFish (+10.5 test return) and a solid improvement on StarPilot (+3.7), but ***Figure 4.6*** confirms this: UCB-DrAC's generalization gap is larger than the PPO baseline's on exactly these games (BigFish 7.58 vs 2.91; FruitBot 7.72 vs 5.48), so the test-return gain comes from a higher training ceiling rather than from closing the gap.. FruitBot is the one game where UCB-DrAC underperforms the PPO baseline on test return (−4.1).

4.2.2 PLR + DrAC (Combined)

Table 4.4 - *PPO baseline vs. PPO + PLR + DrAC: test return by game.*

Game	PPO (Baseline)	PPO + PLR + DrAC	Δ (PLR+DrAC − Baseline)
BigFish	1.5	4.5	+3.0
CoinRun	8.7	7.0	−1.7
FruitBot	23.9	20.6	−3.3

Game	PPO (Baseline)	PPO + PLR + DrAC	Δ (PLR+DrAC − Baseline)
Miner	9.7	5.7	−4.0
Ninja	6.0	5.8	−0.2
StarPilot	27.6	20.4	−7.2

As shown in ***Table 4.4***, on raw test return, PLR + DrAC underperforms the PPO baseline on four of the six games, and underperforms UCB-DrAC on every game in ***Table 4.3***. ***Figure 4.6***, however, shows that PLR + DrAC's generalization gap is near zero or negative on every game, including a negative, better-than-train gap on BigFish and StarPilot, the smallest and most consistent gap of any of the four conditions. This produces a tension worth flagging explicitly for Chapter 5: PLR + DrAC is the most effective technique at closing the generalization gap, but UCB-DrAC achieves the highest absolute test return. Whether the smallest gap or the highest test return is the more important criterion is a judgement call that Section 5.3 addresses directly rather than leaving implicit.

4.2.3 Aggregate Comparison

Figure 4.7 shows the rliable IQM with 95% stratified bootstrap confidence intervals across all 6 games with completed runs, for each of the four conditions.

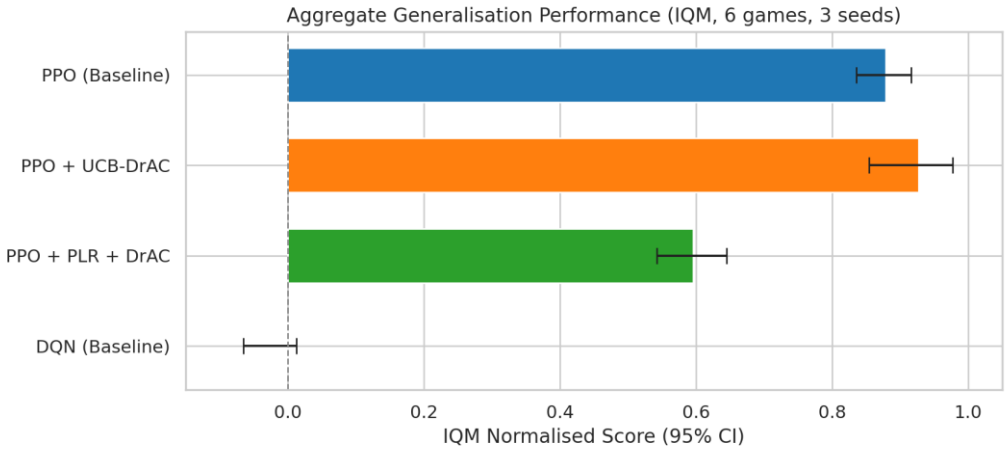

Figure 4.7 - *Aggregate generalization performance: IQM of normalized test return (0 = random policy, 1 = best method per game) across the 6 completed games, with 95% bootstrap confidence intervals*

Figure 4.7 shows that UCB-DrAC has the highest IQM (0.927, 95% CI [0.854, 0.977]; PPO baseline 0.879, CI [0.836, 0.916]), but its confidence interval

overlaps the PPO baseline's, so the improvement is modest and not statistically decisive at three seeds. PLR + DrAC's IQM (0.596, CI [0.542, 0.645]) is lower than UCB-DrAC's but has a visibly tighter confidence interval than the PPO baseline, consistent with the lower per-game variance suggested by its smaller generalization gaps in ***Figure 4.6***. DQN's IQM confidence interval includes zero, indicating no statistically credible aggregate improvement (or degradation) relative to the worst-performing reference point in the normalization.

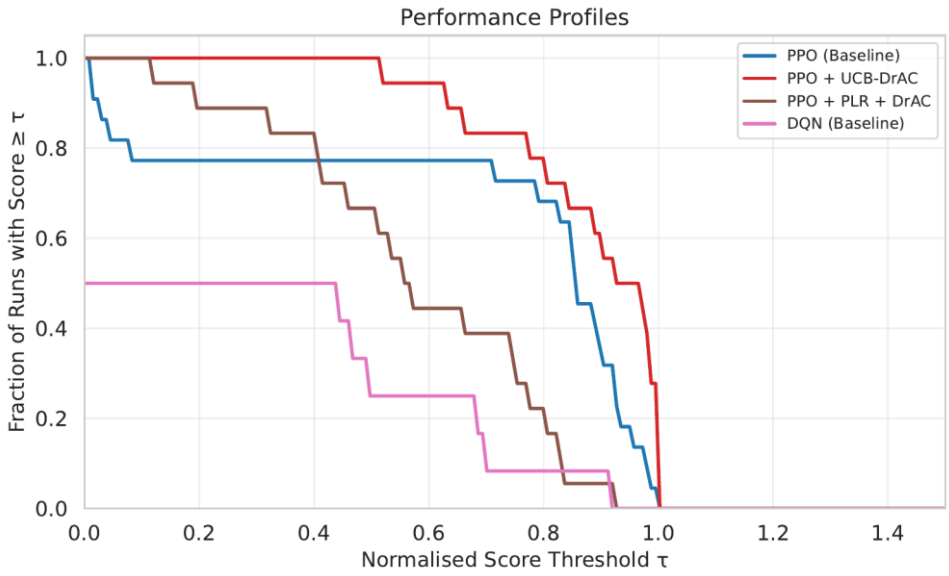

Figure 4.8 - *Performance profiles: fraction of (game, seed) runs achieving at least a given normalized score threshold τ, for each method.*

As shown in ***Figure 4.8***, UCB-DrAC's profile dominates the other three methods across almost the entire threshold range, meaning it is not just better on average but better-or-equal on a large majority of individual game/seed combinations. PPO and PLR + DrAC's profiles cross the exact crossover threshold requires the performance-profile data points, which were not exported in this snapshot, which would be a profile-level signature of the same gap-vs-ceiling tension noted in Section 4.2.2. DQN's profile is lowest and flattest, consistent with its negative aggregate IQM.

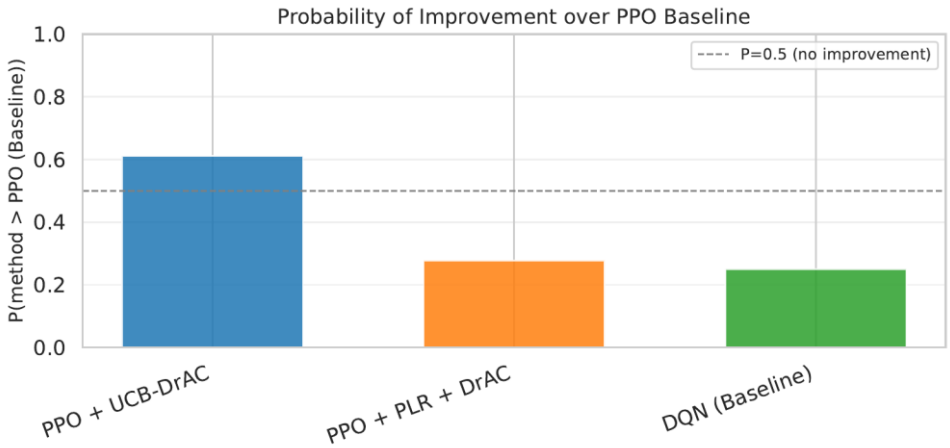

Figure 4.9 - *Probability that each method outperforms the PPO baseline on a randomly drawn (game, seed) pair, with bootstrap confidence intervals. The dashed line at P = 0.5 marks no improvement.*

As shown in ***Figure 4.9***, PPO + UCB-DrAC is above 0.5***,*** at approximately 0.6. This indicates that UCB-DrAC outperforms the baseline on a clear majority of paired comparisons. The corresponding probabilities for PLR + DrAC and DQN against the baseline are both below 0.5, at roughly 0.27 and 0.25 respectively, which at first reads as both techniques underperforming the baseline on a per-comparison basis, apparently in tension with PLR + DrAC's competitive IQM in ***Figure 4.7***. This is exactly the kind of apparent contradiction that pairing aggregate IQM statistics with pairwise probability-of-improvement statistics is meant to surface, and it should be addressed directly rather than cited selectively: it most likely reflects PLR + DrAC reducing variance and worst-case underperformance more than it reflects PLR + DrAC raising typical-case performance. Section 5.1, under RQ3, and Section 5.3 resolve this explicitly using the per-game breakdown in ***Table 4.4***.

Chapter 5: Discussion

5.1 Answering the Research Questions

5.1.1 RQ1: Algorithm Family

RQ1 asked whether the generalization gap differs systematically between policy-based (PPO) and value-based (DQN) methods. The results show a mixed picture rather than the uniform pattern of DQN generalizing worse that was hypothesized in Section 2.1.3: as ***Table 4.2*** shows, PPO outperforms DQN on test return on three of the four shared games, namely CoinRun, Ninja and StarPilot, but underperforms on BigFish, and the aggregate IQM comparison in ***Figure 4.7*** shows DQN's confidence interval spanning zero rather than being uniformly negative. This suggests that the replay-buffer-driven memorization hypothesis is at best a partial explanation. Game-specific factors also matter: BigFish has a very low random-policy baseline of 1.0 and dense, frequent rewards from eating smaller fish, a setting well suited to value-based bootstrapping, which is a plausible reason DQN stays competitive there, offered as a hypothesis rather than a finding given only four games and three seeds. The comparison rests on only the four-game representative subset of BigFish, CoinRun, Ninja and StarPilot, at three seeds, which is a weak basis for any general claim about algorithm families, and it is flagged as a limitation in Section 6.2.

5.1.2 RQ2: Data Augmentation

RQ2 asked to what extent augmentation-based techniques reduce the generalization gap. UCB-DrAC produced the largest absolute test-return improvements of any technique tested (***Table 4.3***, ***Figure 4.7***) and attained the highest aggregate IQM. The improvement over the PPO baseline is, however, modest rather than decisive: the 95% confidence intervals of the two methods overlap, so at three seeds the difference is not statistically conclusive. The per-game generalization-gap data in ***Figure 4.6*** further complicate any simple narrative that augmentation works, because UCB-DrAC's gap is not uniformly smaller than the baseline's; on BigFish and FruitBot it is in fact larger, at 7.58 against 2.91 and 7.72 against 5.48, which means UCB-DrAC's improvement in test return comes at least partly from a higher training ceiling rather than from closing the train/test gap itself. In the final scope, RAD and DrAC alone were not

run, so the progression from RAD to DrAC to UCB-DrAC is answered using only UCB-DrAC against the PPO baseline, and the intermediate steps are deferred to future work in Section 6.3.

5.1.3 RQ3: Curriculum Learning versus Augmentation

RQ3 asked whether PLR addresses a different failure mode from augmentation and whether the two combine. PLR + DrAC achieves the smallest and most consistent generalization gap of any tested condition, near zero or negative on every game in ***Figure 4.6***, while UCB-DrAC achieves the highest absolute test return. This is evidence that the two techniques do functionally different things: PLR + DrAC's contribution looks like genuine gap closure, with training and test performance converging, whereas UCB-DrAC's contribution looks more like raising the agent's overall capability ceiling. That distinction is consistent with the theoretical contrast drawn in Section 2.3.2, namely that curriculum changes the experience distribution while augmentation changes the experience representation. Because the standalone PLR condition was not run, RQ3 is answered provisionally: PLR + DrAC's distinctive small-gap, low-return signature cannot be cleanly attributed to PLR, to DrAC, or to their interaction, and PLR's isolated effect is inferred from the difference between PLR + DrAC and UCB-DrAC and flagged as a limitation in Section 6.2.

5.2 Why Do Agents Fail to Generalize?

Synthesizing across the baseline and the technique comparisons, this thesis considers four candidate, non-exclusive failure modes that contribute to the Procgen generalization gap, each targeted by a different part of the experimental design:

- ***Visual memorization:*** the agent relies on superficial visual features such as background colour, texture or exact sprite positions that correlate with reward during training but do not transfer. This is addressed by data augmentation, that is DrAC and UCB-DrAC, and is evidenced here by UCB-DrAC's large improvement over the baseline on BigFish (Section 4.2.1), the game whose difficulty is most plausibly visual.
- ***Distribution memorization:*** the agent overfits to the specific distribution of level layouts it happens to be trained on, independent of visual appearance. This is addressed by curriculum learning, that is PLR, and is

evidenced by PLR + DrAC's small, consistent generalization gap (Section 4.2.2) relative to its more middling absolute test return.

- ***Capacity-dependent overfitting:*** generalization failure scales with network capacity, since a sufficiently large network can memorize training levels exactly while a smaller network is structurally prevented from doing so, regardless of any explicit regularization. This mode was not tested, because the encoder-capacity ablation was not run, so it remains a hypothesis rather than an evidenced claim (Section 6.3).
- ***Algorithm-family effects:*** the credit-assignment and sample-reuse mechanics of the underlying RL algorithm itself, such as DQN's replay buffer against PPO's single-use rollouts, independently affect generalization, separately from any explicit technique applied on top. This is evidenced, with the caveats noted under RQ1 above, by the PPO/DQN comparison (Section 4.1.2).

On the available evidence there is partial support for visual memorization (UCB-DrAC's large BigFish gain) and for algorithm-family effects (the PPO/DQN contrast); distribution memorization is only suggestively supported (PLR + DrAC's small gaps, confounded by the missing PLR-only condition); and capacity-dependent overfitting is untested (no encoder ablation). No result in this study directly contradicts any of the four hypotheses.

5.3 The Complementarity of Augmentation and Curriculum

Section ***4.2.2*** surfaced an apparent tension: PLR + DrAC has the smallest generalization gap of any condition tested, yet underperforms both the PPO baseline and UCB-DrAC on raw test return on most games, and has a below-0.5 probability of improvement over the PPO baseline (***Figure 4.9***) despite a competitive aggregate IQM (***Figure 4.7***). This is not necessarily a contradiction. A method can simultaneously (a) reduce variance and worst-case failure (which raises its IQM, a robust statistic relatively insensitive to the upper tail) and (b) underperform a higher-variance, higher-ceiling method on a plurality of individual paired comparisons (which lowers its probability of improvement, a statistic sensitive to exactly that ordering). A full per-seed, per-game distributional check requires the raw per-seed export, which is not available in this snapshot; the interpretation is therefore offered as the reading most consistent with the

aggregate (IQM) and pairwise (probability-of-improvement) statistics rather than a verified one.

If this interpretation holds, the practical implication is that PLR + DrAC and UCB-DrAC are not strictly substitutable: a practitioner who cares primarily about worst-case reliability (e.g. deploying an agent in a setting where occasional catastrophic failure is unacceptable) should prefer PLR + DrAC's tighter gap, while a practitioner who cares primarily about peak achievable performance should prefer UCB-DrAC. For the record, a PLR + UCB-DrAC condition was not run: the experiments here combine PLR with fixed-crop DrAC, not with the UCB bandit. Whether the bandit-augmented combination would capture both benefits is left as future work (Section 6.3).

5.4 Practical Recommendations

Distilled into actionable guidance for a practitioner training an RL agent intended to generalize to unseen procedurally generated content:

- If maximizing absolute test performance is the goal and some training instability/variance is acceptable, prefer UCB-DrAC over a fixed single augmentation or no augmentation at all.
- If minimizing worst-case generalization failure (a tight, near-zero train/test gap) is the goal, prefer a PLR-based curriculum over augmentation alone, even at some cost to peak performance.
- No encoder-capacity recommendation can be made: the encoder ablation was not run (future work, Section 6.3).
- No training-diversity recommendation can be made: the num-levels sweep was not run (future work, Section 6.3).

5.5 Threats to Validity

5.5.1 Internal Validity

Hyperparameters are fixed across all games rather than tuned per game (Section 3.1.3), a deliberate methodological choice to keep the cross-method comparison fair. This means that absolute performance for any single game may be lower than a per-game-tuned result reported elsewhere in the literature, so comparisons should be read as relative (between methods, under matched conditions) rather than as new state-of-the-art absolute numbers. Seed variance

is non-trivial in several games (visible in the wide confidence intervals in ***Figures 4.6 and 4.7***), and three seeds per condition, while consistent with common practice in this literature, is a small sample for bootstrap estimation; the clearest consequence is that the UCB-DrAC versus PPO improvement, though positive on the point estimate, has overlapping confidence intervals and so is not statistically decisive.

5.5.2 External Validity

All experiments use 2D, pixel-observation, discrete-action environments from a single benchmark (Procgen, easy mode). Findings about which generalization technique works best, and why, may not transfer to 3D environments, continuous control, partially observed settings, or even Procgen's own hard distribution mode, all of which are explicitly out of scope (Section 6.2) and flagged as future work (Section 6.3).

5.5.3 Construct Validity

The generalization gap as defined (Section 2.4) is sensitive to the choice of reward normalization and evaluation protocol details (Section 3.2.3, 3.5.1): a different normalization scheme or evaluation episode count could shift absolute gap values, though it should not be expected to reverse the qualitative ranking between methods given the consistent application of the same protocol to all conditions. The [0,1] per-game score normalization used for IQM (Section 3.5.2) defines “best” as the best-performing method actually tested, not a theoretical optimum, so IQM values are only meaningful relative to the specific method set compared here and would change if a stronger or weaker method were added to or removed from the comparison.

Chapter 6: Conclusion

6.1 Summary of Contributions

The thesis set out three contributions in Section 1.3,each restated here against what the experimental evidence actually demonstrated. Regarding systematic comparison, UCB-DrAC attained the highest IQM across the six Procgen games studied at three seeds, reaching 0.927, yet its confidence interval overlapped substantially with that of the PPO baseline at 0.879, leaving the aggregate improvement statistically indecisive. Regarding the curriculum-versus-augmentation question, the two strategies produced clearly distinct signatures over the same six games: PPO + PLR + DrAC achieved a near-zero generalization gap but only moderate absolute test return, whereas UCB-DrAC sustained larger generalization gaps alongside higher absolute test return (Section 4.2, Section 5.3). Finally, evidence-based practical recommendations drawn from this comparison are presented in Section 5.4.

6.2 Limitations

This thesis explicitly does not address the following, each a candidate direction for future work (Section 6.3). To begin with, no 3D environments were used during training. All experiments use Procgen's 2D, top-down or side-scrolling games. Secondly, no continuous action spaces were utilized with all environments using small discrete action sets. Thirdly, agents were evaluated zero-shot on unseen levels with no test-time adaptation, so techniques such as MAML-style fast adaptation at test time are out of scope, meaning that no meta-learning approaches were utilized. Also, no population-based training or other hyperparameter-adaptive methods were used with hyperparameters are fixed across the entire experimental matrix (Section 3.1.3). Most importantly, the executed scope is limited to six of the sixteen Procgen games, at three seeds, only at easy distribution mode with no ablation studies and no standalone PLR run. PLR's isolated effect is therefore inferred, not measured.

6.3 Future Work

The most immediate directions for future works are those that would complete the experimental matrix as originally designed. Isolating PLR's standalone contribution, without the DrAC component, is the most pressing of these, since the current study can only infer PLR's effect from the (PLR + DrAC) − (UCB-DrAC) difference. A paired plain (uniform) DrAC condition would serve the same diagnostic purpose on the augmentation side. An ablation isolating which individual transformation each game benefits from would clarify how much of UCB-DrAC's gain is attributable to the bandit selection mechanism rather than to augmentation per se, and this can be cross-checked against the per-game selection frequencies already logged to Weights & Biases. A training-diversity sweep varying the number of training levels would further reveal whether the

techniques shift the saturation point of the generalization curve or merely its ceiling. Two specific results merit dedicated follow-up if they survive a larger study: DQN's unexpected competitiveness on BigFish and the tension between IQM and probability-of-improvement for PLR + DrAC identified in Section 5.3.

A second cluster of open questions concerns the breadth of the evaluation setting. The current study is confined to six of Procgen's sixteen games at easy distribution mode; expanding to the full suite and to the hard distribution would test whether the patterns reported here hold under greater visual and structural diversity. Applying the same techniques to 3D procedurally generated environments — for instance, MineRL-style domains — would probe whether the visual-memorization versus distribution-memorization distinction identified in Section 5.2 persists when observation and action spaces are substantially richer. Multi-task generalization, in which a single agent is trained across several Procgen games simultaneously, would introduce a further dimension by asking whether cross-game transfer interacts with the within-game generalization gap studied here.

A third line of inquiry sits at the algorithmic and architectural level. ACCEL and other adversarial auto-curriculum extensions of PLR adapt the level generator itself rather than only the replay-sampling distribution, and represent a natural next step in the curriculum direction. On the architecture side, testing both substantially larger encoders — ResNet-50-scale CNNs and Vision Transformers — and deliberately undersized ones would quantify how network capacity trades off against overfitting in this regime, completing the encoder-capacity ablation that was designed but not executed. A DrAC component ablation separating the actor-only, critic-only, and full consistency-loss contributions would complement this architectural analysis and clarify whether both consistency terms are necessary or whether one dominates.

Chapter 7: References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., & Bellemare, M. (2021). Deep reinforcement learning at the edge of the statistical precipice. Advances in Neural Information Processing Systems (NeurIPS), 34. arXiv:2108.13264.

Cobbe, K., Hesse, C., Hilton, J., & Schulman, J. (2020). Leveraging procedural generation to benchmark reinforcement learning. Proceedings of the 37th International Conference on Machine Learning (ICML), PMLR 119, 2048–2056.

Dennis, M., Jaques, N., Vinyals, O., Foerster, J., Russell, S., Critch, A., & Levine, S. (2020). Emergent complexity and zero-shot transfer via unsupervised environment design. Advances in Neural Information Processing Systems (NeurIPS), 33. arXiv:2012.02096.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., & Kavukcuoglu, K. (2018). IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. Proceedings of the 35th International Conference on Machine Learning (ICML), PMLR 80, 1407–1416.

Farebrother, J., Machado, M. C., & Bowling, M. (2018). Generalization and regularization in DQN. arXiv preprint arXiv:1810.00123.

Jiang, M., Grefenstette, E., & Rocktäschel, T. (2021). Prioritized level replay. Proceedings of the 38th International Conference on Machine Learning (ICML), PMLR 139, 4940–4950.

Kirk, R., Zhang, A., Grefenstette, E., & Rocktäschel, T. (2023). A survey of zero-shot generalisation in deep reinforcement learning. Journal of Artificial Intelligence Research, 76, 201–264.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., & Srinivas, A. (2020). Reinforcement learning with augmented data. Advances in Neural Information Processing Systems (NeurIPS), 33.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533.

Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., & Song, D. (2018). Assessing generalization in deep reinforcement learning. arXiv preprint arXiv:1810.12282.

Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E., & Rocktäschel, T. (2022). Evolving curricula with regret-based environment design. Proceedings of the 39th International Conference on Machine Learning (ICML), PMLR 162, 17473–17498.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., & Fergus, R. (2021). Automatic data augmentation for generalization in reinforcement learning. Advances in Neural Information Processing Systems (NeurIPS), 34.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. Proceedings of the 4th International Conference on Learning Representations (ICLR).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Biewald, L. (2020). Experiment tracking with Weights & Biases. Software available from wandb.com.

Chevalier-Boisvert, M., Dai, B., Towers, M., Perez-Vicente, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S., & Terry, J. (2023). Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. Advances in Neural Information Processing Systems (NeurIPS), 36. arXiv:2306.13831.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2019). Quantifying generalization in reinforcement learning. Proceedings of the 36th International Conference on Machine Learning (ICML), PMLR 97, 1282–1289.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., & Araújo, J. G. (2022). CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. Journal of Machine Learning Research, 23(274), 1–18.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI).

Igl, M., Ciosek, K., Li, Y., Tschiatschek, S., Zhang, C., Devlin, S., & Hofmann, K. (2019). Generalization in reinforcement learning with selective noise injection and information bottleneck. Advances in Neural Information Processing Systems (NeurIPS), 32. arXiv:1910.12911.

Lee, K., Lee, K., Shin, J., & Lee, H. (2020). Network randomization: A simple technique for generalization in deep reinforcement learning. International Conference on Learning Representations (ICLR). arXiv:1910.05396.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489.

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

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 575(7782), 350–354.

Wang, K., Kang, B., Shao, J., & Feng, J. (2020). Improving generalization in reinforcement learning with mixture regularization. Advances in Neural Information Processing Systems (NeurIPS), 33. arXiv:2010.10814.

Weng, J., Lin, M., Huang, S., Liu, B., Makoviichuk, D., Makoviychuk, V., Liu, Z., Song, Y., Luo, T., Jiang, Y., Xu, Z., & Yan, S. (2022). EnvPool: A highly parallel reinforcement learning environment execution engine. Advances in Neural Information Processing Systems (NeurIPS), 35. arXiv:2206.10558.

Zhang, C., Vinyals, O., Munos, R., & Bengio, S. (2018). A study on overfitting in deep reinforcement learning. arXiv preprint arXiv:1804.06893.

Appendix A: Full Per-Game Results Tables

This appendix reproduces the full per-game results, namely train return, test return and generalization gap, for every experimental condition, as raw tables that complement the summary tables in Chapter 4. All values are three-seed means with standard deviations, computed from the deduplicated per-seed results.

A.1 PPO (Baseline)

Table 7.1 *- PPO IMPALA-CNN baseline: per-game train return, test return, and generalization gap for the six games studied (mean ± standard deviation across 3 seeds).*

Game	Train Return	Test Return	Gen. Gap
BigFish	4.50 ± 1.72	1.59 ± 0.49	2.91 ± 1.24
CoinRun	9.73 ± 0.06	8.67 ± 0.15	1.07 ± 0.21
FruitBot	29.34 ± 0.62	23.86 ± 0.63	5.48 ± 0.66
Miner	12.83 ± 0.07	9.71 ± 0.48	3.12 ± 0.45
Ninja	8.10 ± 0.36	6.03 ± 0.50	2.07 ± 0.40
StarPilot	29.68 ± 2.26	27.90 ± 2.93	1.79 ± 0.89

A.2 PPO + UCB-DrAC

Table 7.2 *- PPO + UCB-DrAC: per-game train return, test return, and generalization gap for the six games studied (mean ± standard deviation across 3 seeds).*

Game	Train Return	Test Return	Gen. Gap
BigFish	19.54 ± 3.85	11.96 ± 3.46	7.58 ± 1.27
CoinRun	9.57 ± 0.06	8.87 ± 0.21	0.70 ± 0.17
FruitBot	27.51 ± 1.09	19.79 ± 4.97	7.72 ± 4.18
Miner	12.64 ± 0.36	10.25 ± 0.81	2.39 ± 0.57
Ninja	8.40 ± 0.46	5.83 ± 0.55	2.57 ± 0.15

Game	Train Return	Test Return	Gen. Gap
StarPilot	31.28 ± 2.61	31.27 ± 0.29	0.01 ± 2.88

A.3 PPO + PLR + DrAC

Table 7.3 *- PPO + PLR + DrAC: per-game train return, test return, and generalization gap for the six games studied (mean ± standard deviation across 3 seeds).*

Game	Train Return	Test Return	Gen. Gap
BigFish	4.31 ± 1.15	4.50 ± 2.45	−0.19 ± 1.95
CoinRun	7.37 ± 0.97	7.03 ± 0.70	0.33 ± 0.29
FruitBot	21.65 ± 3.45	20.64 ± 2.36	1.01 ± 1.49
Miner	5.89 ± 0.67	5.68 ± 0.70	0.21 ± 1.36
Ninja	5.97 ± 0.42	5.80 ± 0.10	0.17 ± 0.45
StarPilot	19.35 ± 2.66	20.44 ± 3.02	−1.09 ± 2.15

A.4 DQN (Baseline)

The DQN baseline targets the four-game representative subset by design (Section 3.1.2), so this table has four data rows, not by missing data.

Table 7.4 *- DQN baseline: per-game train return, test return, and generalization gap, four-game representative subset (mean ± standard deviation across 3 seeds).*

Game	Train Return	Test Return	Gen. Gap
BigFish	9.82 ± 0.63	7.39 ± 0.33	2.44 ± 0.30
CoinRun	2.33 ± 0.61	2.10 ± 1.11	0.23 ± 0.50
Ninja	0.70 ± 0.66	0.47 ± 0.64	0.23 ± 0.32
StarPilot	24.50 ± 2.84	24.68 ± 3.95	−0.18 ± 1.95

Appendix B: Hyperparameter Configurations

B.1 PPO Configuration

Reproduced from the project's training configuration file, used unchanged across all PPO-family conditions, namely the PPO baseline, UCB-DrAC and PLR + DrAC:

```
total_timesteps:                                                25_000_000
num_envs:                                                               64
num_steps:                                                             256
num_epochs:                                                              3
num_minibatches:                                                         8
learning_rate:                                                       5e-4
gamma:                                                              0.999
gae_lambda:                                                          0.95
clip_coef:                                                            0.2
ent_coef:                                                            0.01
vf_coef:                                                              0.5
max_grad_norm:                                                        0.5
num_levels:                                                           200
distribution_mode:                                                  easy
encoder:                                                          impala
checkpoint_freq:                                                 500_000
eval_freq:                                                       500_000
eval_episodes: 100
```

B.2 DQN Configuration

The DQN baseline run on starpilot, bigfish, coinrun and ninja:

B.3 Augmentation and Curriculum Hyperparameters

UCB-DrAC and PLR each introduce additional hyperparameters beyond the base PPO configuration in Section B.1. The as-run values are:

UCB-DrAC bandit exploration constant (c in the UCB1 score): 0.1

DrAC regularization weight (KL policy-consistency and MSE value-consistency loss coefficients): a single coefficient, aug_coef = 0.1, weights both the KL (actor) and MSE (critic) consistency terms equally; drac_mode = full

PLR replay probability \u03c1: 0.5

PLR staleness coefficient and scoring temperature \u03b2: staleness coefficient 0.1; scoring temperature β = 0.1; scoring rule = L1 value loss

Augmentations available to DrAC and UCB-DrAC (src/augmentations.py): random crop, colour jitter, grayscale, cutout, horizontal flip, ±15° rotation, and random convolution. PLR + DrAC uses a fixed random crop; UCB-DrAC selects among all seven via the UCB1 bandit. RAD was not run as a separate condition.

Appendix C: Additional Figures

C.1 Additional Per-Game Overfitting Curves

Figures 4.1 – 4.4 in Chapter 4 show four representative games, namely CoinRun, StarPilot, BigFish and Ninja. The two remaining games with completed runs, FruitBot and Miner, are included here for completeness.

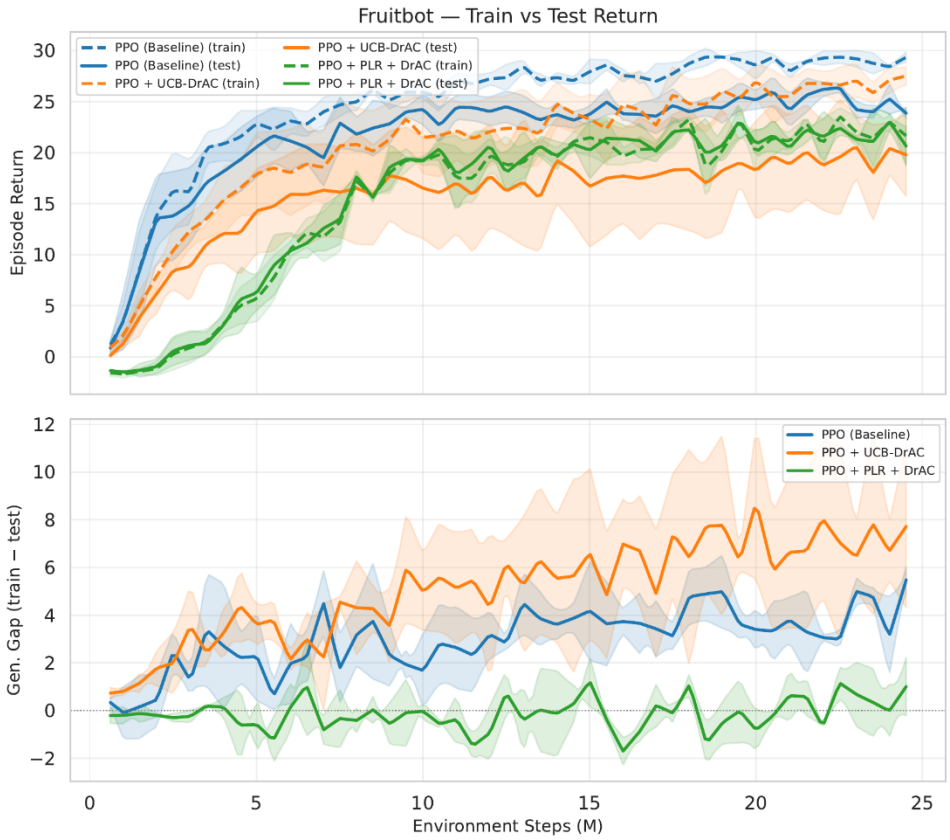

Figure 7.1 *- FruitBot: train and test episodic return (top) and generalization gap (bottom) vs. environment steps, all available methods*

Figure 7.2 *- Miner: train and test episodic return (top) and generalization gap (bottom) vs. environment steps, all available methods.*