
What Does a ProcGen Generalization Gap Measure? Action Rules, Convergence, and the Missing Random Floor

Abhisek Keshari

Independent researcher

abhisek.keshari12@gmail.com

Abstract

A generalization gap in reinforcement learning (return on training levels minus return on held-out levels) is usually reported without a reference point. We argue that the missing reference is a measured random floor: the return of a uniform-random policy on the same levels under the same harness. On ProcGen, the floor changes what several standard numbers mean. On identical checkpoints and levels across eight environments, switching between sampled and greedy (argmax) test-time actions moves held-out return in both directions, and greedy evaluation takes three environments to or below the floor: in miner, the sampled policy scores $4.9\times$ the floor on held-out levels while its argmax scores below it. Raw policy entropy places seven of eight environments short of convergence, but 35–65% of that entropy is spread across actions with identical effects; after merging them, one to three remain short, and against the floor only heist has learned nothing that transfers. An audit of twelve prior ProcGen codebases finds that all eleven with held-out evaluation sample test-time actions for their policy-gradient agents, nine by default rather than explicit choice, and six report running in-loop averages rather than evaluating a fixed checkpoint. Applied to our own case study, the same checks grade down a statistically significant encoder effect and rule out a within-encoder train-vs-test CKA statistic. We recommend that every reported gap state its action rule, use a matched and seeded protocol, and report the random floor on both level sets.

1 Introduction

A generalization gap in reinforcement learning (RL)—return on training levels minus return on held-out levels—is the field’s headline transfer metric [Cobbe et al., 2019, Kirk et al., 2023], and it is almost always reported without a reference point. This paper argues that the missing reference is a measured random floor: the return of a uniform-random policy on the same levels, under the same harness. It costs minutes to measure, and on the benchmark we study it changes what several standard numbers mean.

The setting is ProcGen [Cobbe et al., 2020], a standard benchmark for generalization in deep RL. Each ProcGen environment procedurally generates *levels*: episodes whose layout, textures, and object placement vary under a seeded generator. An agent is trained on a fixed set of levels (here 200) and evaluated on held-out levels it has never interacted with (here 100), drawn from the same generator. Algorithms, architectures, and regularizers are compared by their gaps, and representation-level explanations are attached to the winners.

The first thing the floor exposes is that the test-time action rule decides which policy gets measured. An agent’s action can be *sampled* from its action distribution or chosen greedily as the most probable action (argmax). For a converged policy the two nearly coincide; for one that has not converged

they are different policies, and the difference is not a scalar inflation. On identical checkpoints and identical levels, switching rules moves held-out return in both directions across eight environments (Section 5.1). In miner, sampling from the trained policy scores $4.9\times$ the floor on held-out levels, while its argmax—the rule our own main tables use—scores below the floor. A policy that demonstrably transfers is measured as worse than chance.

The second is convergence. How much a gap says about transfer depends on whether the policy has learned a training-level behavior to transfer. A common proxy is policy entropy against the maximum of $\ln 15 = 2.708$ over ProcGen’s 15 discrete actions; by that measure a standard 8M-step budget leaves seven of eight environments at 74–95% of maximum. But ProcGen games have between 4 and 11 functionally distinct actions, and 35–65% of the measured entropy is spread across actions with identical effects (Appendix D). Against the floor, one environment, heist, has learned nothing that transfers: its sampled held-out return sits at the floor at 8M and at or marginally below it after a threefold budget extension, while its training-level return reaches twice the floor. Entropy remains useful as a cheap screen. It agrees with the floor at the extremes and over-calls in the middle, most clearly on bossfight, which it flags as non-converged and whose policy scores clearly above a near-zero floor ($z \approx 2.6$ – 2.7). High entropy is not proof that training failed, and a stochastic policy is a legitimate object of evaluation.

Neither check is standard. Auditing the released evaluation code of twelve prior ProcGen codebases, we find that all eleven that evaluate on held-out levels sample the test-time actions of their policy-gradient agents. Nine do so through a default function argument (`deterministic=False`) that no call site overrides or through the training rollout’s own sampling call; two set the rule explicitly. Two codebases also evaluate value-based agents, which act greedily (Impoola’s DQN) or ϵ -greedily (EDE’s own method). Six read test numbers from a held-out rollout running inside the training loop, averaged over episodes that span policy updates, rather than evaluating a fixed checkpoint (Section 4). Sampling on both level sets is a matched protocol, and by the analysis above it measures the policy that was trained; what the audit documents is a rule that in most codebases is inherited rather than set, so that its consequences for the reported gap go unexamined.

We apply the same checks to our own case study (two encoder architectures, all else fixed) and to our own harness. A statistically significant encoder effect on heist turns out to compare two below-floor held-out returns; an unseeded evaluation-level draw moved a p -value across the significance boundary; and a representation statistic we had computed, within-encoder train-vs-test CKA, proves invalid on disjoint observation sets.

Our contributions:

1. A measured random floor as the reference point for generalization gaps, cheap to compute, reported for eight ProcGen environments on both level sets.
2. Evidence that the test-time action rule decides which policy is measured: on identical checkpoints and levels the two rules disagree in both directions, and greedy evaluation takes three of eight environments to or below the floor, including one whose sampled policy scores $4.9\times$ it.
3. A recalibration of entropy-based convergence checks: effective action counts per game, merged-action entropy, and a behavioral check against the floor, which together reduce seven of eight environments short of convergence to one to three, and identify the one environment whose policy has learned nothing that transfers.
4. A twelve-codebase audit of ProcGen evaluation practice, verified against pinned commits to the file and line, including a correction of our own earlier classification.
5. A self-audit: an encoder effect graded down by the floor, a level-draw variance source in our harness, and a 2×2 control with a closed-form null showing that within-encoder train-vs-test CKA is invalid on disjoint inputs.

2 Related Work

Evaluation methodology in RL. Henderson et al. [2018] document reproducibility failures caused by underspecified evaluation procedures; Agarwal et al. [2021] argue for statistically rigorous reporting (interquartile means, stratified bootstrap intervals) given RL’s small seed counts. Those works ask

whether results are *reported* rigorously. We ask a question one step earlier: whether the two numbers being differenced mean what they appear to, which requires knowing which policy produced them and where chance lies.

Why the default exists. Prior ProcGen protocols were designed largely for *ranking*. The original benchmark [Cobbe et al., 2020] and the NeurIPS 2020 ProcGen competition [Mohanty et al., 2021] average many episodes for a stable leaderboard, a setting in which stochastic action selection is internally consistent across submissions. The default is thus a defensible design choice, and by Section 5.1 a sound one for policies that have not converged; what was lost when it was reused is the reason for it. When a single train-minus-test difference becomes a paper’s headline number, which rule produced it, whether it describes one checkpoint, and where it sits relative to chance all bear directly on what it means.

Policy convergence. Beukman et al. [2026] document that PPO training frequently stagnates far short of a task-solving policy; several recent PPO variants target exactly this stagnation [Rahman and Xue, 2022, Lixandru, 2024, Rahman, 2025, Xie et al., 2025]. Our increment is to show that stagnation confounds the *measurement* rather than just the performance. A gap computed on a stagnated policy is confounded with incomplete optimization, and is correspondingly weaker evidence about transfer.

Encoders and representation diagnostics. The IMPALA CNN [Espeholt et al., 2018] is the standard ProcGen encoder; feature pyramid networks [Lin et al., 2017] are a multi-scale design imported from object detection, and VSOP-3D [Jesson and Jiang, 2024] reports ProcGen gains from combining architectural scale-up with an algorithmic change [Jesson et al., 2024]. For representation analysis we use linear centered kernel alignment (CKA) [Kornblith et al., 2019], a similarity measure between two networks’ internal representations, with the caveats of Davari et al. [2022], and the dormant-neuron ratio [Sokar et al., 2023], the fraction of network units that have effectively stopped activating.

3 Method

3.1 Encoder architectures

Two encoders, identical except for one design choice. Both share the same three-block IMPALA convolutional backbone. The baseline flattens the final feature map and projects it to a 256-dimensional representation. The *multiscale FPN* variant additionally fuses intermediate feature maps from all three blocks, top-down, before projecting to the same 256-dimensional space (521k vs. 622k parameters; architectural details in Appendix F). The policy and value heads, the RL algorithm, and every hyperparameter are identical across the two conditions, so any downstream difference is attributable to the encoder.

3.2 Training setup

We train agents with PPO, a standard on-policy deep RL algorithm: the agent alternates between collecting experience with its current policy (a *rollout*) and updating the policy on that experience. The 8M runs use 16 parallel environment copies, and the 25M runs and the heist entropy-coefficient retrains use 256 (the CleanRL ProcGen default is 64); all use 200 training levels and an otherwise standard configuration (Appendix F); each condition (encoder $\times$ environment) is repeated with 3 random seeds—independent training runs differing only in random initialization. At 25M, starpilot and heist use 9 and 6 seeds per encoder; the additional seeds were trained later with an identical configuration. The baseline budget is 8M environment steps across eight environments; a budget extension trains three environments to 25M steps.

3.3 Evaluation protocol

Three properties of an evaluation protocol determine what a reported gap measures.

Which policy. At evaluation time an action can be sampled from the policy’s action distribution or chosen greedily as the highest-probability action (argmax). Sampling evaluates the policy that was trained. Argmax evaluates a deterministic policy derived from it, which coincides with the trained policy only when the action distribution is sharply peaked. A harness can describe itself as deterministic while sampling at every step, so the rule has to be checked in code, not read from a configuration name.

Which checkpoint. A commonly logged metric is the running mean return of recent episodes collected during training. Such a number averages over several policy versions and describes no single checkpoint, and comparing it against a separately measured return would also conflate a difference in procedure with the train/test shift.

Relative to what. A return is interpretable only against a reference. We measure a uniform-random floor on both level sets with the same harness, level draws, and episode accounting as the trained policies (Appendix D). Cobbe et al. [2020] normalize ProcGen returns with fixed per-game constants. The floor used here differs in being measured rather than fixed: it is taken under the evaluating harness, level draws and episode accounting, on both level sets, so it moves with the protocol it calibrates.

Our main tables evaluate both level sets identically with greedy argmax, 128 episodes per measurement at the 25M budget and 64 at 8M, at the final checkpoint, with an episode burn-in and complete rollout windows. Section 5.1 re-evaluates every 8M checkpoint under both rules on seeded level draws, against the floor. Given what that section finds, greedy numbers for environments short of convergence describe the argmax policy, and we read them that way.

3.4 Convergence checks

How much a generalization gap says about transfer depends on whether the policy has learned a training-level behavior in the first place. A gap measured on a policy that has not is confounded with incomplete optimization: harder to read as transfer, not uninterpretable. We use two checks. The *behavioral check* compares held-out and training-level return, under the policy’s own sampled actions, with the random floor; it needs no threshold, normalizer, or action-space assumption. The *screen* is final-checkpoint policy entropy relative to the maximum-entropy uniform policy ($\ln 15 = 2.708$), in three tiers: **converged** (entropy < 2.0), **partial** ([2.0, 2.3]), **non-converged** (> 2.3). The screen is cheap and routinely logged; the behavioral check is what a tier should be read against. Entropy alone cannot deliver a verdict: a high-entropy policy may be a legitimately stochastic policy and a legitimate object of evaluation, and a low-entropy policy may still perform poorly. Entropy is the mean over the train- and test-level evaluation passes; at 25M we report heist under both that convention and the test-level value alone, because the two differ enough to change its label (Appendix B).

The entropy bonus is not the only configuration choice that moves the measurement. Retraining heist with $c_H = 0.001$ instead of the default 0.01 gives a terminal entropy of 2.113 ± 0.038 (78.0% of maximum; $n=3$ seeds per encoder, 8M), against 2.565 (94.7%) for the default runs. Those retrains, however, used 256 parallel environments rather than the 16 of the default 8M runs (Appendix F), so the difference cannot be attributed to c_H ; it shows only that the screen’s value depends on the training configuration, which is one more reason to read it against the behavioral check.

The two checks agree at the extremes and disagree in the middle (Table 3). Starpilot is converged by entropy and far above the floor; heist is the least converged by entropy and the only environment whose sampled held-out return is at the floor. Between them the screen over-calls. Bossfight is non-converged by entropy yet its sampled policy scores clearly above the floor on held-out levels (1.38 vs. 0.09, $z \approx 2.6$), and dodgeball, also non-converged, is above its floor (1.12 vs. 0.47, $z \approx 3.0$). Part of the over-call is the action space: entropy over ProcGen’s 15 raw actions counts probability spread across actions with identical effects as indecision, and 35–65% of each environment’s raw entropy is of that kind (Appendix D). Merging equivalent actions before computing entropy leaves one environment short of the converged band (heist) when equivalence is determined state by state, and three (dodgeball, bossfight, heist) under a single partition per game; these counts depend on expressing the 2.0 and 2.3 cuts as fractions of the maximum, which is our choice. We keep raw-action entropy for the tiers because it is what practitioners log, and read a partial or non-converged tier as a prompt to run the behavioral check, not as a finding that the policy has not learned.

Two robustness checks address whether the entropy cut point is load-bearing. First, sweeping the converged/partial boundary across [1.8, 2.2] leaves the two extreme calls unchanged (starpilot is converged and heist non-converged at every value), while the intermediate environments near 2.0 move with the cut point, which is precisely why they carry the *partial* label rather than a verdict (Appendix B). This sweep bounds the borderline calls only: starpilot (1.799) and heist (2.565) are the extremes of the observed range, so no interior cut point could reclassify them, and their stability under the sweep follows from the data layout rather than from robustness. Second, a diagnostic that

never thresholds the final entropy (the total *collapse* of entropy from its initial plateau, combined with training-set return) ranks environments consistently with the tiers (Spearman $\rho = 0.79$; Appendix C); it shares the terminal-entropy trajectory as an input, so it is a second reading rather than a fully independent one.

3.5 Statistical analysis

Encoder and train-vs-test comparisons use Welch’s t -test. Where we pool across independent draws (seeds), per-draw two-sided p -values are combined with Fisher’s method, with Stouffer’s Z on the one-sided p -values as a directional cross-check; confidence intervals are 95% percentile bootstrap (10^4 resamples). Per-environment pooled results, and one instructive case where the omnibus and directional combiners disagree, are in Appendix A.

4 Field-Wide Audit

To document how ProcGen generalization numbers are produced in practice, we audited the released evaluation code of twelve prior ProcGen codebases spanning ICML, NeurIPS, RLC, and arXiv from 2019 to 2025.

What we looked for. For each repository we located the code path that produces the paper’s reported test numbers. We then classified two binary properties: (i) *stochastic evaluation*, whether test-time actions are sampled from the policy distribution rather than chosen by greedy argmax; and (ii) *in-loop test returns*, whether reported test returns are read from a held-out rollout that runs concurrently inside the training loop—stepped alongside training, never reset between updates, and reported as a running mean over the most recent completed episodes, which therefore span several policy versions—rather than from a dedicated evaluation of a fixed checkpoint. Property (ii) is deliberately narrower than it may sound: in every codebase we mark, the held-out environments or workers contribute no gradient, so the test returns are *not* returns from the rollout the policy is trained on. Classifications were made against a pinned commit, and the evidencing file and line for every entry are listed in Appendix E so the classification is reproducible against the precise code state. For codebases with an explicit evaluation script, the cited line is the test-time action-selection call; for the in-loop codebases, the cited lines are where the held-out environment or worker is created and where its running return is reported.

What we found. Eleven of twelve codebases perform held-out evaluation; *all eleven* evaluate their policy-gradient agents stochastically, and six of those additionally report in-loop test returns (Table 1). UCB-DrAC [Raileanu et al., 2021] is the most explicit instance, passing `deterministic=False` at `test.py` line 42. Impoola [Trumpf et al., 2025] also sets the rule explicitly, through a documented configuration option: its PPO evaluation samples and its DQN evaluation is greedy, as its paper states. EDE [Jiang et al., 2023] is itself a value-based method: its released code samples when evaluating its PPO baseline, which is the row we classify, while EDE’s own agent is evaluated ϵ -greedily with $\epsilon = 0.05$, the standard DQN evaluation protocol. For a value-based agent the greedy action is the trained policy, so neither case bears on the sampled-versus-argmax analysis of Section 5.1, which concerns stochastic policies. VSOP and VSOP-3D [Jesson et al., 2024, Jesson and Jiang, 2024] step a separate held-out environment once per training step and report the mean of its last 100 episodes.

A classification we checked and rejected. These six codebases are easily read as *rollout-mode*—reading test returns off the on-policy training rollout, a train-vs-test procedure mismatch—and our own first pass classified them that way. Checking each against its pinned commit shows that classification is wrong. In mixreg, the ProcGen baseline, and IBAC-SNI the test numbers come from a dedicated MPI test worker whose gradient is weighted to zero; in RAD, VSOP, and VSOP-3D they come from a separate held-out environment whose transitions are discarded. In all six the train number comes from the matching in-loop procedure on the training levels, so train and test are measured the same way. What remains is the weaker property defined above: the test number is a sampled-action running average taken during training, not a measurement of the final policy. Our cited location for mixreg was also wrong: it pointed at an evaluation runner the training script never enables. The twelfth, PPG [Cobbe et al., 2021], trains on the full level distribution and performs no held-out evaluation in its released code, so neither classification applies (n/a).

Table 1: Code audit of twelve prior ProcGen codebases. Eleven perform held-out evaluation and all eleven use stochastic action selection at test time for their policy-gradient agents (Stoch.); six of those additionally report test returns from a held-out rollout running inside the training loop, averaged over episodes that span policy updates, rather than evaluating a fixed checkpoint (In-loop). In none of the six is the policy trained on that rollout. PPG performs no held-out evaluation in its released code (n/a). *UCB-DrAC passes `deterministic=False` explicitly at `test.py` line 42. †Impoola sets the rule through a configuration option: its PPO evaluation samples and its DQN evaluation is greedy. ‡The row classifies EDE’s PPO baseline; EDE’s own value-based agent is evaluated ϵ -greedily ($\epsilon = 0.05$). Commit hashes and file/line evidence for every row: Appendix E.

Codebase	Venue	Stoch.	In-loop
VSOP-3D [Jesson and Jiang, 2024]	arXiv’24	✓	✓
VSOP [Jesson et al., 2024]	ICML’24	✓	✓
ProcGen [Cobbe et al., 2020]	ICML’20	✓	✓
mixreg [Wang et al., 2020]	NeurIPS’20	✓	✓
RAD [Laskin et al., 2020]	NeurIPS’20	✓	✓
IBAC-SNI [Igl et al., 2019]	NeurIPS’19	✓	✓
DAAC [Raileanu and Fergus, 2021]	ICML’21	✓	–
UCB-DrAC [Raileanu et al., 2021]	NeurIPS’21	✓*	–
PLR [Jiang et al., 2021]	ICML’21	✓	–
EDE [Jiang et al., 2023]	NeurIPS’23	✓‡	–
Impoola [Trumpp et al., 2025]	RLC’25	✓†	–
PPG [Cobbe et al., 2021]	ICML’21	n/a	n/a

Provenance. Nine of the eleven set no rule at the point of evaluation. Three (DAAC, PLR, and EDE’s PPO baseline) inherit sampling through a default argument that no call site overrides, and the six in-loop codebases sample through the training rollout’s own sampling call, which has no deterministic path. In these nine, nothing in the evaluation code selects the rule or states a reason for it. That is how the practice became field-wide: an evaluation default propagated through copied harness code. The two exceptions show that the rule can be set deliberately: UCB-DrAC passes `deterministic=False` at the call site, and Impoola exposes the rule as a configuration option, sampling for PPO and acting greedily for DQN. By the analysis of Section 5.1, the default happens to be the better of the two rules for policies that have not converged; the concern is that where it is inherited, its consequences for the reported gap go unexamined.

Scope of the claim. We audited released evaluation code, not results as reported in papers, and we do not claim any specific published number is wrong. The claim is about what the procedure can support. A gap whose train and test numbers come from different procedures cannot distinguish a protocol difference from a distributional one; we found no such case in the audit, but the hazard is general. An in-loop gap is matched, but it averages over the policies visited during training rather than measuring any one of them. A gap under matched sampled evaluation measures the trained policy (Section 5.1); what it lacks, as reported, is a reference point.

5 Results

5.1 The action rule decides which policy is measured

Table 2 evaluates the same 48 checkpoints (8M steps; eight environments $\times$ two encoders $\times$ three seeds) on the same held-out levels under both rules, 128 episodes per run, alongside a uniform-random policy on identical level draws. None of what follows is a scalar inflation.

One checkpoint, two answers. Miner shows the whole effect in one pair of numbers. Sampling from the trained 8M policies scores 5.69 ± 0.33 on held-out levels, $4.9\times$ the random floor of 1.17 ± 0.11 ; the argmax of the same weights scores 0.64 ± 0.10 , below the floor in all six runs. The policy has learned something that transfers, and greedy evaluation—the rule our own main tables use—reports it as worse than chance. Coinrun shows the same pattern more weakly: sampled 5.82, above its floor of 3.28; greedy 3.55, indistinguishable from it.

The rule’s effect has no fixed sign. Sampled return exceeds greedy return in miner, coinrun and heist (3.05 vs. 0.51). Greedy exceeds sampled in bossfight (2.42 vs. 1.38, all six runs) and fruitbot

Table 2: Held-out (test-level) return of the same 8M checkpoints under greedy (argmax) and sampled action selection, with a uniform-random policy on identical level draws. Mean $\pm$ SEM over six runs (two encoders $\times$ three seeds), 128 episodes per run; the random floor is mean $\pm$ SEM over three evaluation seeds. The last two columns compare each rule with the floor (z from both SEMs; below: $z < -2$, at: $|z| \leq 2$, above: $z > 2$). Greedy values are regenerated on seeded level draws and differ from Table 13 (64 episodes, unrecorded draws) by at most about one SEM.

Env	Random	Greedy	Sampled	Greedy	Sampled
starpilot	1.67 ± 0.03	17.97 ± 1.69	18.36 ± 1.12	above	above
fruitbot	-2.66 ± 0.25	25.63 ± 0.41	22.74 ± 0.43	above	above
bigfish	0.84 ± 0.05	3.12 ± 0.89	2.97 ± 0.89	above	above
coinrun	3.28 ± 0.12	3.55 ± 0.71	5.82 ± 0.28	at	above
miner	1.17 ± 0.11	0.64 ± 0.10	5.69 ± 0.33	below	above
dodgeball	0.47 ± 0.05	0.86 ± 0.05	1.12 ± 0.21	above	above
bossfight	0.09 ± 0.05	2.42 ± 0.85	1.38 ± 0.50	above	above
heist	3.15 ± 0.07	0.51 ± 0.19	3.05 ± 0.22	below	at

(25.63 vs. 22.74). The remaining three are within noise, among them starpilot, the one environment converged by entropy (17.97 vs. 18.36). There is no single correction factor that converts one rule’s numbers into the other’s.

Why argmax can fall below chance. Argmax of a policy that has not converged is a different policy. It commits every state to whichever action narrowly leads, and where that preference is arbitrary the committed behavior can be worse than choosing at random—for example, repeating one failing trajectory where random play would eventually stumble onto the goal (a plausible mechanism, not one we isolate). A second candidate, specific to ProcGen’s action space, is that argmax over raw actions can choose against the policy’s own preference when the probability of one behavior is split across functionally equivalent actions (Appendix D); in miner, where the three leftward and three rightward moves each collapse to one, this can happen at any step. We do not test it here.

When the sampled number is at chance, it is accurate. Heist is the other case. Its sampled held-out return (3.05 ± 0.22) is indistinguishable from the floor (3.15 ± 0.07). Heist pays 0 or 10 per level, so the sampled policy completes about 30% of held-out levels, random play about 31%, and the greedy policy about 5%. Unlike miner’s, heist’s sampled number accurately reports a policy that transfers nothing beyond random play. Read as a ratio, sampled over greedy— $8.3\times$ for IMPALA (2.58 vs. 0.31) and $5.0\times$ for FPN (3.52 vs. 0.70)—it looks like stochastic inflation; read against the floor, it is greedy deflation below chance.

What this does to the gap. Applying one rule to both level sets is a *matched* measurement, and our audit found no codebase that takes train and test numbers from structurally different procedures (Section 4). The concern is not a mismatch. It is that the rule decides which policy is measured, and for a policy that has not converged the two candidates differ in kind, so the gaps differ too. At 8M the sampled gap exceeds the greedy gap in heist (1.46 vs. 0.76) and miner (1.60 vs. 0.55), and is similar or smaller elsewhere. At 25M on heist ($n=6$ seeds per encoder; Table 15) the sampled gap is $1.69\times$ the greedy one (4.36 ± 0.19 vs. 2.58 ± 0.26 ; Welch $t=5.47$, $p=2.4 \times 10^{-5}$), and the floor says what each gap is made of. Under sampling, training-level return is twice the floor (7.17 vs. 3.54) and held-out return is at or marginally below it (2.81 ± 0.16 vs. 3.15 ± 0.07 , $z = -1.95$): a policy that has learned its training levels and transfers nothing. Under greedy selection, training-level return is at the floor (3.38) and held-out return is below it (0.79): a difference between chance and sub-chance. Neither gap is interpretable without the floor, and the sampled one is the more faithful description of the policy that was trained.

5.2 Convergence across eight environments

Table 3 sets the entropy screen beside the behavioral check at 8M. By raw entropy only starpilot converges (1.799, 66.4% of maximum); four environments are partial, and dodgeball (89.5%), bossfight (87.9%), and heist (94.7%) are non-converged. Against the floor, seven of eight sampled policies are above chance on held-out levels, and only heist is at it. Merging functionally equivalent actions moves the screen toward the behavioral reading: state by state, only heist remains short of the converged band.

Table 3: Convergence at 8M, pooled across both encoders and 3 seeds each. Entropy is the mean of train- and test-level policy entropy over raw actions; maximum = $\ln 15 = 2.708$. Merged: tier after summing the probabilities of functionally equivalent actions, with equivalence determined per state / by one partition per game (Appendix D; cuts at the same fractions of the maximum). Floor: sampled held-out return against the random floor (Table 2). Gap is the pooled deterministic train–test return difference. C/P = converged/partial.

Environment	Entropy	% max	Tier (raw)	Merged	Floor	Gap
starpilot	1.799	66.4	converged	C / C	above	−0.99
fruitbot	2.119	78.3	partial	C / C	above	0.72
bigfish	2.073	76.5	partial	C / C	above	2.56
coinrun	2.130	78.7	partial	C / C	above	0.21
miner	2.009	74.2	partial	C / C	above	0.39
dodgeball	2.423	89.5	non-converged	C / P	above	0.49
bossfight	2.379	87.9	non-converged	C / P	above	0.38
heist	2.565	94.7	non-converged	P / P	at	0.52

Bossfight is the sharpest disagreement. The screen calls it non-converged; its sampled and greedy policies both sit above the near-zero floor on held-out levels ($z \approx 2.6$ and 2.7). We report this as a misclassification by the screen. A raw-entropy tier says that the policy’s distribution over raw actions is broad, and for bossfight that breadth sits on top of a policy that has plainly learned; it does not say that the policy has failed to learn. Heist, whose gap looks most interesting under sampled evaluation, is the one environment on which both checks agree.

5.3 Budget extension to 25M steps

Is non-convergence just insufficient training? We extend starpilot, fruitbot, and heist to 25M steps (Figure 1). These runs also use 256 parallel environments rather than 16 (Section 7), so the extension changes batch size and the number of policy updates along with the step budget. Starpilot’s entropy crosses the threshold at roughly 5–7M steps and continues to 55–63% of maximum. Fruitbot’s raw training-log entropy sits at 80–82% of maximum across the 17M additional steps and *plateaus* (against 78.3% at 8M, an evaluation-pass value on a different convention). But at 8M, 65% of fruitbot’s raw entropy is spread over equivalent actions and its policy is far above the floor at 8M, so the plateau is a limit of the screen, not evidence that fruitbot failed to learn. Heist is the informative case. Its entropy declines modestly (train-level 77–83%, test-level 86–87%). Behaviorally, three times the budget lifts its sampled training-level return to twice the floor (7.17 vs. 3.54; Table 15) while its sampled held-out return stays at or marginally below the floor (2.81 ± 0.16 vs. 3.15 ± 0.07 , $z = -1.95$; the floor is measured on the same level sets but a different level draw). The additional environment steps raised training-level return, and none of that improvement transfers. Training length cannot stand in for a convergence check, and for heist the check that matters is behavioral.

Convergence does not, however, make the gap stable. Starpilot is the one environment converged by the screen, and its gap changes sign between budgets: -0.99 [$-1.66, -0.35$] at 8M against $+3.11$ [$1.27, 4.81$] at 25M (Table 5), scoring held-out levels slightly *above* training levels early and clearly below them later. Convergence makes a gap easier to read; it does not make the reading budget-independent. The two budgets also differ in parallelism, so the sign change cannot be attributed to budget alone.

5.4 A third variance source, in our own harness

The matched-protocol requirement extends to a place we did not initially look: the random draw of the evaluation level set itself. ProcGen samples the evaluated subset of held-out levels without a fixed seed, so re-evaluating *identical* checkpoints under an otherwise identical protocol instantiates a different level subset each run. On heist this moves the encoder test-return p -value from 0.04 to 0.13 across draws, crossing the conventional significance boundary, while the gap difference survives both draws ($p = 0.004$ – 0.011). We report the range rather than a combined p -value across draws: Fisher’s method assumes independence, and re-evaluations of the *same* checkpoints share the trained weights, so pooling over them would be anticonservative (pooling across seeds, which have disjoint RNG streams, is unaffected). We therefore seed the evaluation draw and report the gap as robust but the

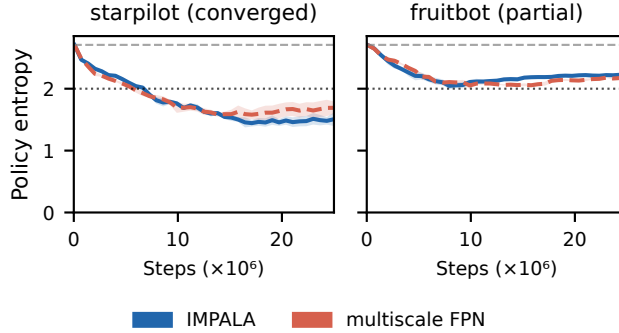


Figure 1: Policy entropy over raw actions during training at the 25M budget, starpilot and fruitbot, measured from per-iteration training logs (mean over the original three seeds per encoder; band is ± 1 SEM). Starpilot crosses the entropy threshold and keeps descending; fruitbot’s raw entropy plateaus above it, largely over functionally equivalent actions (Section 5.3).

test-return difference as not; a third, seeded draw of the same checkpoints agrees (gap $p = 0.0023$, test return $p = 0.12$). We flag this deliberately. It is the same class of unexamined protocol variance we audit in others, it was in our own harness, and it changed a conclusion. An evaluation protocol is not fully specified until every random draw in it is either fixed or marginalized over.

5.5 Encoder comparison, convergence-conditional

The case study asks whether a multi-scale encoder generalizes better than the standard one. How far the answer can be read as a statement about generalization depends on convergence and on where the returns sit relative to the floor (Table 4).

At convergence: null. On starpilot at 25M ($n=9$ seeds), test return ($p=0.728$) and gap ($p=0.481$) are both null. The capacity diagnostic agrees: dormant-neuron ratios overlap at every threshold (full tables in Appendix F).

Outside convergence: an apparent effect, graded down. On heist at 25M ($n=6$), FPN’s gap is significantly smaller (2.11 ± 0.21 vs. 3.40 ± 0.27 ; $p = 0.004$ – 0.011 across the two unseeded evaluation draws, and $p = 0.0023$ on a seeded third draw). The floor says what those numbers are. They are greedy returns: both encoders’ held-out returns (0.47 and 0.89) sit far below the held-out floor of 3.15, and their training-level returns (3.87 and 2.99) straddle the training floor of 3.54. The encoder difference compares two below-chance measurements of argmax policies, and the sampled policy they stand in for transfers nothing beyond random play (Section 5.3). It is real as a measurement and weak evidence about generalization. Decomposing it shows why caution is warranted even at face value. Of the 1.29-point gap difference, 0.87 (68%) comes from *lower FPN train return* rather than better transfer, and the test-return difference is not robust across evaluation draws (p ranges across the boundary, 0.04–0.13). Cross-encoder CKA is markedly highest in the converged environment (0.757/0.783 train/test, against 0.518–0.678 elsewhere), consistent with convergence driving distinct architectures toward a shared solution, though tier is confounded with environment identity across three environments, so we claim only the converged-vs-rest contrast.

A representation diagnostic that does not work. Within-encoder train-vs-test CKA is a natural way to ask how level-specific a representation is: compute each encoder’s representation of train-level and of held-out-level observations and compare the two. It does not work. Linear CKA compares two representation matrices row by row, so row i of each must be the same input, and no valid correspondence exists between two disjoint observation sets; here 1024 train-level observations are set against 1024 *different* test-level ones. The value is not noise—it sits 8 to 97 standard deviations above its row-permutation null (per-seed range, starpilot and fruitbot at 8M)—but a 2×2 control identifies what it responds to: holding the row alignment while removing the domain shift entirely (train-vs-train) returns the same value as the train-vs-test comparison, and wherever the representation has appreciable effective rank, shuffling one side’s rows collapses both, by 4.6 – $7.2\times$ on fruitbot at 8M. It reads the episode-timestep alignment shared by the two observation batches, not the level

Table 4: Final-checkpoint results at 25M steps, deterministic evaluation, mean $\pm$ SEM (starpilot $n=9$, heist $n=6$, fruitbot $n=3$ seeds per encoder), 128 episodes per measurement. † gap difference significant on each evaluation draw separately ($p = 0.004\text{--}0.011$ unseeded, 0.0023 seeded; not pooled, Section 5.4). Tier is the raw-entropy screen (Section 3.4); heist’s held-out returns here are below the random floor of 3.15.

Env	Encoder	Train	Test	Gap	Tier
starpilot	IMPALA	33.14 ± 1.38	29.36 ± 1.99	3.79 ± 1.16	converged
starpilot	FPN	30.81 ± 2.41	28.39 ± 1.89	2.42 ± 1.49	converged
fruitbot	IMPALA	29.52 ± 0.27	22.99 ± 0.39	6.53 ± 0.51	partial
fruitbot	FPN	28.10 ± 0.08	21.15 ± 0.56	6.95 ± 0.50	partial
heist	IMPALA	3.87 ± 0.30	0.47 ± 0.12	$3.40 \pm 0.27^\dagger$	non-conv.
heist	FPN	2.99 ± 0.30	0.89 ± 0.13	$2.11 \pm 0.21^\dagger$	non-conv.

set, and its permutation null is exactly $\sqrt{\text{PR}_X \text{PR}_Y} / (n-1)$ in the two participation ratios, so the between-encoder offset tracks effective rank. Read naively, the statistic says FPN is $3.44\times$ less train-specific than IMPALA on fruitbot at 8M; in units of each side’s own null the ratio is $1.14\times$, and seed by seed it straddles one (Appendix G). The cross-encoder rows are unaffected: they compare two encoders on *identical* observations, which is correctly paired. We report the diagnostic and its failure, rather than omitting it, because the failure generalizes beyond ProcGen: any train-vs-test representation comparison computed with a row-paired similarity measure on disjoint inputs is exposed to it.

6 Discussion

Report the rule and the floor. The recommendation that follows from Section 5.1 is not “use sampling”; it is to state the action rule and report a measured random floor on both level sets. What distinguishes the two rules is which policy they evaluate: sampling evaluates the policy that was trained, argmax a deterministic policy derived from it. The two coincide at convergence—starpilot’s rules agree within noise (17.97 vs. 18.36)—so argmax is safe there, and unsafe exactly where policies have not converged: every floor crossing in Table 2 occurs in an environment the entropy screen flags as partial or non-converged. Argmax remains reasonable when the deterministic policy is itself the object of interest, or as a second number beside the sampled one; its danger is being read as the trained policy’s competence. Sampling is not self-interpreting either: heist’s sampled numbers are accurate, and only the floor shows that they are accurate reports of chance.

What a trustworthy gap requires. Stated as a procedure: evaluate a fixed checkpoint rather than a running in-loop average; state the action rule and apply it identically to both level sets; seed (or marginalize over) every random draw in the evaluation pipeline, including the level-subset draw; measure the random floor on both level sets with the same harness; and check convergence behaviorally, against that floor, using entropy as a screen. Each step is cheap, and the floor costs minutes; none is standard.

Where it matters most. The rules diverge, and returns sit nearest the floor, in environments the entropy screen flags, which are also where apparent gaps look largest and most interesting. A gap there needs its floor more than anywhere else.

Reliability, stated precisely. This is the paper’s standing position on convergence. Our claim concerns whether a number supports the inference drawn from it, not that all such numbers are noise, and not that a high-entropy policy is illegitimate: a stochastic policy is a real policy and can be a proper object of evaluation. What convergence governs is whether the train-minus-test *gap* supports an inference about transfer. An environment whose held-out return sits well above the random floor (fruitbot, 25.4 greedy mean across six runs at 8M, against -2.66 for uniform-random play) retains transferable behavior, and its gap is cautiously interpretable whatever its entropy tier; a policy at the floor (heist, sampled 3.05 against 3.15) supports much less. The checks grade interpretability; they do not zero it out.

7 Limitations

Seed counts. Most conditions use $n=3$ seeds ($n=6$ heist, $n=9$ starpilot at 25M), so the encoder comparison has limited statistical power. The two claim classes differ here. The action-rule and convergence findings rest on effects that do not turn on seed count: floor crossings in all six runs of heist and miner, a $4.9\times$ sampled-to-floor margin on miner, and a twelve-codebase audit. The encoder comparison does turn on seed count, and we present it as a case study rather than a finding.

The floor is one reference. A uniform-random policy is the simplest reference and calibrates only the lower end; a stronger reference, such as a scripted policy or one trained to convergence, would calibrate the upper end, and we do not report one. The floor’s level draws match the 8M re-evaluation exactly but not the 25M heist comparison, which used a different seeded draw from the same level sets. Random-weight guessing [Oller et al., 2020], which evaluates untrained networks rather than uniform actions, would give a structured lower reference that we have not measured.

Our own tables use argmax. Tables 13 and 4 and the gaps in Table 3 were measured with greedy selection before the analysis of Section 5.1. For environments short of convergence they describe the argmax policy; apart from heist (Table 15) we have not re-run the 25M budget under sampling.

One threshold, one suite, one algorithm. The entropy screen uses a single threshold, swept but not derived, on one benchmark suite with one RL algorithm (PPO), over raw rather than functionally distinct actions. The suite is eight of ProcGen’s sixteen games, in the easy distribution mode only. The action-equivalence classes are measured from the rendered observation over a five-step horizon, and could miss effects that surface later. Success-rate stability over training would be a natural further check and requires per-step return logs we did not export.

Parallel environments. The 8M runs use 16 parallel environments and the 25M runs 256, against the CleanRL default of 64. Each PPO update therefore sees a smaller batch at 8M and a larger one at 25M, and with 256-step rollouts the 25M runs make about 381 policy updates against about 1953 at 8M. The budget extension is thus an extension in environment steps rather than optimization steps, and it changes batch size along with budget. Beukman et al. [2026] show that PPO stagnation can be relieved by scaling the number of parallel environments; our 25M runs used four times the default and heist and fruitbot still did not converge, but the 8M entropy levels may partly reflect the smaller batch. The action-rule and floor findings describe the checkpoints as trained and do not depend on why a policy has not converged; the convergence counts and the 8M-to-25M comparisons do.

Audit scope. The audit covers released evaluation code at pinned commits, not results as reported in papers; papers may have used unreleased evaluation scripts.

Figures. The 8M entropy values are tabulated (Table 3) rather than plotted, because per-step traces for those runs were never exported; Figure 1 comes from per-iteration training logs. Every figure and tabulated number here is measured.

8 Conclusion

A generalization gap is a measurement, and without a reference point it is hard to say what it measures. In ProcGen, a measured random floor shows that the test-time action rule decides which policy gets measured—a policy that transfers at $4.9\times$ chance can be scored below chance by its own argmax—and that an entropy screen over-calls non-convergence: by the behavioral check, one environment of eight has a policy that has learned nothing that transfers. The field’s policy-gradient evaluation code samples actions, in nine of eleven codebases by default rather than explicit setting, and in half the audited codebases reports running in-loop averages. We recommend three cheap additions to every reported gap: state the action rule, match and seed the protocol, and report the random floor on both level sets. Applied to our own study, they graded down our one significant result and ruled out a natural representation statistic.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

- Michael Beukman, Khimya Khetarpal, Zeyu Zheng, Will Dabney, Jakob Foerster, Michael Dennis, and Clare Lyle. Preventing learning stagnation in PPO by scaling to 1 million parallel environments. *arXiv preprint arXiv:2603.06009*, 2026.
- Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, ICML’19. PMLR, 2019.
- Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, ICML’20. PMLR, 2020.
- Karl W. Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *International Conference on Machine Learning (ICML)*, 2021.
- MohammadReza Davari, Stefan Horoi, Amine Natik, Guillaume Lajoie, Guy Wolf, and Eugene Belilovsky. Reliability of CKA as a similarity measure in deep learning. *arXiv preprint arXiv:2210.16156*, 2022.
- Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr 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. In *Proceedings of the 35th International Conference on Machine Learning*, ICML’18. PMLR, 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.
- Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschieschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- Andrew Jesson and Yiding Jiang. Improving generalization on the ProcGen benchmark with simple architectural changes and scale. *arXiv preprint arXiv:2410.10905*, 2024.
- Andrew Jesson, Chris Lu, Gunshi Gupta, Nicolas Beltran-Velez, Angelos Filos, Jakob N. Foerster, and Yarin Gal. ReLU to the rescue: Improve your on-policy actor-critic with positive advantages. In *Proceedings of the 41st International Conference on Machine Learning*, ICML’24, pages 21577–21605. PMLR, 2024.
- Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning (ICML)*, 2021.
- Yiding Jiang, J. Zico Kolter, and Roberta Raileanu. On the importance of exploration for generalization in reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 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.
- Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning (ICML)*, pages 3519–3529, 2019.
- Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

- Andrei Lixandru. Proximal policy optimization with adaptive exploration. *arXiv preprint arXiv:2405.04664*, 2024.
- Sharada Mohanty, Jyotish Poonganam, Adrien Gaidon, Andrey Kolobov, Blake Wulfe, et al. Measuring sample efficiency and generalization in reinforcement learning benchmarks: NeurIPS 2020 procgen benchmark. *arXiv preprint arXiv:2103.15332*, 2021.
- Declan Oller, Tobias Glasmachers, and Giuseppe Cuccu. Analyzing reinforcement learning benchmarks with random weight guessing. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020. arXiv:2004.07707.
- Ben Rahman. PPO-BR: Dual-signal entropy-reward adaptation for trust region policy optimization. *arXiv preprint arXiv:2505.17714*, 2025.
- Md Masudur Rahman and Yexiang Xue. Robust policy optimization in deep reinforcement learning. *arXiv preprint arXiv:2212.07536*, 2022.
- Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2021.
- Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, 2021.
- Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. PMLR, 2023.
- Raphael Trumpp, Ansgar Schäfftlein, Mirco Theile, and Marco Caccamo. Impoola: The power of average pooling for image-based deep reinforcement learning. *Reinforcement Learning Journal (RLJ)*, 2025. arXiv:2503.05546.
- Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- Zhengpeng Xie, Qiang Zhang, Fan Yang, Marco Hutter, and Renjing Xu. Simple policy optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 68813–68824. PMLR, 2025.

A Pooled Significance and Confidence Intervals

For each environment we treat every independently trained and evaluated run (one seed and encoder, scored on its own sampled held-out level subset) as one draw, and test the train-vs-test difference within a draw with a Welch t -test from that draw’s episode-level statistics. Per-draw p -values are combined across draws with Fisher’s method (a χ^2 omnibus combiner); Stouffer’s Z on the one-sided (train > test) p -values is a directional, sign-preserving cross-check. Both combiners assume independent draws, which holds because seeds use disjoint RNG streams and independently sampled level subsets. Confidence intervals are 95% percentile bootstrap (10^4 resamples) on the mean per-draw gap.

Per-environment results are in Table 5. The three statistics answer different questions. Fisher and Stouffer both combine the per-draw Welch tests: Fisher is sensitive to *any* draw showing a difference, and Stouffer additionally requires the draws to agree in direction. The bootstrap CI does not combine per-draw tests at all; it is computed on the mean of the per-draw gaps. They disagree in two rows, in opposite directions. On fruitbot at 8M, Fisher $p=0.015$ but Stouffer $Z=1.1$ with a CI spanning zero: the gap is not directionally reliable, consistent with that environment’s partial convergence. On starpilot at 8M the pattern inverts: Fisher $p=0.98$ alongside a CI of $[-1.66, -0.35]$ that excludes zero. The six per-draw two-sided p -values are 0.93, 0.94, 0.76, 0.75, 0.46 and 0.58; combined, they give a large Fisher p , while Stouffer’s Z (-0.79) agrees in finding no directional evidence. The bootstrap CI is computed on the mean of per-draw gaps and is therefore sensitive to a small effect

that is stable across draws, which is what starpilot at 8M has. The CI is the informative statistic in that row. The tier column is a reminder that significance and interpretability are separate questions: the heist 8M row is significant under both combiners, and heist’s sampled policy sits at the random floor on held-out levels (Section 5.2), so it is weak evidence about transfer.

Table 5: Generalization gap (train — test) significance per environment, pooled across independent draws (seeds $\times$ encoders).

Env	Budget	Tier	n	Gap [95% CI]	Fisher p	Stouffer Z
starpilot	8M	converged	6	$-0.99 [-1.66, -0.35]$	0.98	-0.79
starpilot	25M	converged	18	$3.11 [1.27, 4.81]$	1.6×10^{-8}	5.29
fruitbot	8M	partial	6	$0.72 [-1.42, 2.44]$	0.015	1.09
fruitbot	25M	partial	6	$6.74 [6.16, 7.29]$	$< 10^{-30}$	13.4
heist	8M	non-converged	6	$0.52 [0.23, 0.86]$	0.018	3.20
heist	25M	non-converged	12	$2.75 [2.29, 3.23]$	$< 10^{-30}$	19.2

B Threshold Sensitivity of the Convergence Tiers

Sweeping the converged/partial boundary across $[1.8, 2.2]$ (Table 6) leaves both headline calls fixed. Starpilot is converged and heist non-converged at every value. Only the borderline environments near 2.0 change tier within the range, as expected.

Table 6: Convergence tier under a swept converged/partial threshold T (8M budget; partial/non-converged cut held at 2.3). C = converged, P = partial, N = non-converged.

Env	Entropy	$T=1.8$	$T=2.0$	$T=2.2$
starpilot	1.799	C	C	C
miner	2.009	P	P	C
bigfish	2.073	P	P	C
fruitbot	2.119	P	P	C
coinrun	2.130	P	P	C
bossfight	2.379	N	N	N
dodgeball	2.423	N	N	N
heist	2.565	N	N	N

At 25M, heist stays outside the converged tier under both entropy conventions used in the paper (Table 7): test-level entropy, the transfer-relevant measure, is 86.5% of maximum (non-converged); the train+test mean used for the 8M tiers is 83.2% (partial). Starpilot is converged under every convention.

Table 7: Heist and starpilot final-checkpoint entropy at 25M under two conventions ($n=6$ heist, $n=9$ starpilot; % of $\ln 15$).

Env	Convention	% max	Tier
starpilot	test-level	60.2	converged
starpilot	train+test mean	60.6	converged
heist	test-level	86.5	non-converged
heist	train+test mean	83.2	partial

C Second Convergence Diagnostic

As a second reading that never thresholds the final entropy value (though it shares the entropy trajectory as an input, so it is not fully independent), we measure the *dynamics* of entropy collapse (total drop from the initial plateau, and terminal entropy) together with training-set return (Table 8). A converged policy collapses far and ends low; a non-converged policy barely collapses and stays high. The ranking agrees with the entropy tiers (Spearman $\rho=0.79$ between converged-ness and total

entropy drop; $\rho=0.55$ against train return): starpilot has the lowest terminal entropy and heist the highest; by total collapse, bigfish and starpilot collapse furthest (0.66 and 0.64) and coinrun and heist least (-0.08 and 0.07).

Table 8: Second convergence diagnostic (8M, pooled over encoders and seeds), ordered by the primary entropy tier. Terminal entropy and entropy drop here are computed from the logged training entropy under the sampled policy on training levels (terminal: mean over the last 25% of each run), and differ from Table 3, which averages greedy evaluation passes on both level sets at the final checkpoint. The entropy-drop column excludes 7 of 48 runs whose logs begin after the first logging interval, since their initial plateau is not recorded.

Env	Tier	Ent. drop	Term. ent.	Train ret.
starpilot	converged	0.64	1.78	16.84
miner	partial	0.41	2.22	1.07
bigfish	partial	0.66	1.89	5.62
fruitbot	partial	0.38	2.10	26.12
coinrun	partial	-0.08	2.31	3.75
bossfight	non-converged	0.26	2.31	2.71
dodgeball	non-converged	0.15	2.37	1.24
heist	non-converged	0.07	2.59	0.83

D Calibration: Random-Policy Floor and Effective Actions

Random-policy floor. Uniform-random actions over the 15 actions, evaluated with the same harness, level sets and episode accounting as the trained policies (3 evaluation seeds $\times$ 128 episodes per level set; mean $\pm$ SEM across seeds). Trained returns are the deterministic 8M returns pooled over both encoders and three seeds (Table 13).

Table 9: Uniform-random floor against trained deterministic returns (8M). “Ratio” is trained test over random test. Ratios are unstable where the random floor is near zero (bossfight) or small (dodgeball); Table 2 gives floor-relative z -values.

Env	Tier	Random train	Random test	Trained train	Trained test	Ratio
starpilot	converged	1.40 ± 0.17	1.67 ± 0.03	16.84	17.82	10.7
fruitbot	partial	-2.43 ± 0.21	-2.66 ± 0.25	26.12	25.40	—
bigfish	partial	0.90 ± 0.04	0.84 ± 0.05	5.62	3.06	3.65
coinrun	partial	2.94 ± 0.37	3.28 ± 0.12	3.75	3.54	1.08
miner	partial	1.24 ± 0.01	1.17 ± 0.11	1.07	0.68	0.58
dodgeball	non-converged	0.57 ± 0.05	0.47 ± 0.05	1.24	0.76	1.61
bossfight	non-converged	0.01 ± 0.01	0.09 ± 0.05	2.71	2.33	25.5
heist	non-converged	3.54 ± 0.16	3.15 ± 0.07	0.83	0.31	0.10

Greedy policies can fall below the random floor; a plausible reason, which we have not isolated, is that argmax removes the variability random play gets for free, so a deterministic policy that has not learned the task can repeat the same failing trajectory.

Effective action counts. To test whether the 15 actions are equally meaningful, we sampled 400 states per game (200 per level set) by random play, cloned each state, applied every action from the identical state, and compared the resulting 64×64 observation and reward over the action and four subsequent no-ops. Replaying the same action from the same state gave identical outcomes in every case, so equal outcomes identify equivalent actions. Counts are not uniform: the number of distinct outcome classes across all sampled states is 11 (starpilot), 10 (dodgeball, bossfight), 9 (bigfish, coinrun, heist), 5 (miner) and 4 (fruitbot); per state the mean is lower still in coinrun (3.8) and miner (4.2), where most actions coincide unless the agent is at a ledge or wall. The special-action keys are inert in most games, and in fruitbot and miner the three leftward and three rightward moves each collapse to one.

Normalizing raw entropy by $\ln k$ instead of $\ln 15$ is not meaningful: raw entropy exceeds $\ln k$ in six of eight games, precisely because mass is spread over equivalent actions. Merging first is the valid

Table 10: Final 8M policy entropy (nats, reproduced from the checkpoints) before and after summing the probabilities of functionally equivalent actions, using each state’s own equivalence classes. “Share” is the fraction of raw entropy spread across equivalent actions.

Env	k (global)	Raw H	Merged H	Share
starpilot	11	1.775	1.133	36%
fruitbot	4	2.108	0.735	65%
bigfish	9	2.102	1.363	35%
coinrun	9	1.978	0.907	54%
miner	5	1.824	0.749	59%
dodgeball	10	2.368	1.538	35%
bossfight	10	2.365	1.204	49%
heist	9	2.522	1.590	37%

comparison. It preserves the extremes—starpilot has the lowest merged entropy relative to its action count and heist the highest—but the raw partial tier (fruitbot, coinrun, miner) owes most of its entropy to equivalent actions, so under a merged measure those environments look more decided than the raw tiers suggest. Their greedy returns do not follow: coinrun’s sits at its random floor and miner’s below it (Table 9), even though both environments’ sampled policies are well above the floor (Table 2); the greedy numbers measure the argmax policy, not the trained one. Reproduced raw entropies are within 0.06 nats of Table 3 for six games and lower for coinrun (−0.15) and miner (−0.19), because the reproduction starts every environment afresh at each checkpoint.

E Audit: Commit Hashes and Evidence Locations

Table 11 lists, for each audited codebase, the exact source commit examined and the file and line evidencing each property, so the classification in Table 1 is reproducible against the precise code state. Where an action-selection call takes a `deterministic` argument, the default-False definition site is noted. For EDE’s own value-based agent, `train_rainbow.py:642` applies an evaluation ϵ whose default of 0.05 is set at `level_replay/dqn_args.py:101`.

Table 11: Commit hash and evidence location for each audited codebase. S = stochastic test evaluation, L = in-loop test return (held-out environment or worker stepped inside the training loop, not trained on). For L rows the location is where the held-out environment or worker is created, then where its running return is reported.

Codebase	Venue	Commit	S	L	Location (file:line)
UCB-DrAC	NeurIPS’21	1fbf2567cca1	✓	–	<code>test.py:42 (deterministic=False)</code>
DAAC	ICML’21	2fe302029428	✓	–	<code>test.py:31 (act(), default False)</code>
PLR	ICML’21	ccecf452ee33	✓	–	<code>test.py:71 (default False, test.py:34)</code>
EDE	NeurIPS’23	89396f548aaa	✓	–	<code>test.py:70 (PPO; default :30); train_rainbow.py:642 (ϵ)</code>
Impoola	RLC’25	1d71110451cc	✓	–	<code>evaluation.py:30; ppo_training.py:47 (F), dqn_training.py:50 (T)</code>
VSOP	ICML’24	7764611bdc98	✓	✓	<code>ppo_procgen.py:368 (probs.sample()); :457 → :691</code>
VSOP-3D	arXiv’24	6eb3ce463b47	✓	✓	<code>vsop_3d_procgen.py:354 (probs.sample()); :440 → :693</code>
mixreg	NeurIPS’20	0800f492491f	✓	✓	<code>train.py:66–72 test worker → ppo2.py:219</code>
RAD	NeurIPS’20	d61a1ac8258a	✓	✓	<code>train.py:77 eval venv → ppo2.py:220</code>
IBAC-SNI	NeurIPS’19	6b3a58bfc23a	✓	✓	<code>policies.py:196 (pd_run.sample()); config.py:198 → ppo2.py:395</code>
ProcGen	ICML’20	1a2ae2194a61	✓	✓	<code>train.py:28 test worker → baselines ppo2.py:201</code>
PPG	ICML’21	7295473f0185	n/a	n/a	no held-out evaluation

F Additional Tables

This appendix collects the full hyperparameter configuration (Table 12), the complete 8M per-encoder breakdown (Table 13), the full CKA results (Table 14), and the dormant-neuron ratios (Table 16).

Encoder details. Both encoders share a three-block IMPALA backbone (channel widths [16, 32, 32]; each block is a 3×3 convolution, stride-2 max-pool, and two residual sub-blocks) on (3, 64, 64) RGB input. The baseline flattens the final $32 \times 8 \times 8$ map and projects to 256 dimensions. The multiscale FPN variant projects each block’s feature map to 32 channels via 1×1 lateral convolutions, fuses them top-down with nearest-neighbor upsampling [Lin et al., 2017], smooths each fused map with a 3×3

convolution, pools to 4×4 , concatenates (1536 dimensions), and projects to the same 256-dimensional space. FPN has fewer parameters (521k vs. 622k) but higher per-forward compute (0.087 vs. 0.061 GFLOPs). Policy head, value head, initialization, and all training hyperparameters are identical.

Table 12: PPO hyperparameters (fixed across both encoders and all environments; the CleanRL ProcGen configuration except for the number of parallel environments, whose default is 64). The 25M runs change the total timesteps and the number of parallel environments; all other values are identical.

Hyperparameter	Value
Parallel environments	16 (8M); 256 (25M, c_H retrain)
Rollout length	256
Total timesteps	8,000,000
Learning rate	5×10^{-4} (no annealing)
Discount factor γ	0.999
GAE λ	0.95
Minibatches per update	8
Update epochs	3
Clip coefficient	0.2
Value loss clipping	enabled
Entropy coefficient	0.01 (0.001 in the c_H retrain)
Value loss coefficient	0.5
Max gradient norm	0.5

Table 13: Final-checkpoint (8M) results, all 8 environments, both encoders. Mean $\pm$ SEM across 3 seeds, deterministic evaluation, matched train/test protocol.

Env	Encoder	Train	Test	Gap
starpilot	IMPALA	18.56 ± 0.34	18.91 ± 0.49	-0.35 ± 0.34
starpilot	FPN	15.11 ± 3.28	16.73 ± 3.17	-1.63 ± 0.38
fruitbot	IMPALA	26.85 ± 1.23	26.87 ± 0.90	-0.02 ± 1.93
fruitbot	FPN	25.39 ± 0.99	23.94 ± 1.57	1.45 ± 1.31
bigfish	IMPALA	2.09 ± 0.10	1.07 ± 0.06	1.02 ± 0.14
bigfish	FPN	9.15 ± 1.90	5.04 ± 0.22	4.10 ± 2.09
coinrun	IMPALA	5.31 ± 0.56	4.48 ± 0.10	0.83 ± 0.60
coinrun	FPN	2.19 ± 0.48	2.60 ± 0.77	-0.42 ± 0.29
miner	IMPALA	1.42 ± 0.30	0.71 ± 0.14	0.71 ± 0.28
miner	FPN	0.72 ± 0.15	0.65 ± 0.22	0.07 ± 0.21
dodgeball	IMPALA	1.26 ± 0.25	0.64 ± 0.05	0.62 ± 0.20
dodgeball	FPN	1.23 ± 0.21	0.88 ± 0.22	0.35 ± 0.41
bossfight	IMPALA	0.96 ± 0.19	1.33 ± 0.17	-0.37 ± 0.29
bossfight	FPN	4.45 ± 0.99	3.32 ± 0.94	1.13 ± 0.46
heist	IMPALA	0.83 ± 0.28	0.21 ± 0.14	0.62 ± 0.33
heist	FPN	0.83 ± 0.05	0.42 ± 0.23	0.42 ± 0.19

Table 14: Linear CKA (mean $\pm$ std over 3 seeds), 256-dimensional final encoder representation, 1024 observations per level set. Cross-encoder rows compare the two architectures on *identical* observations and are correctly paired. The within-encoder train-vs-test rows are shown for transparency but are **not valid measurements**: they pair each observation against an unrelated one, and the controls in Appendix G show they do not measure train-specificity.

Env	Comparison	8M		25M	
		Mean	Std	Mean	Std
starpilot	IMPALA: train vs. test	0.020	0.001	0.022	0.004
starpilot	FPN: train vs. test	0.020	0.007	0.021	0.009
starpilot	cross-encoder train	0.703	0.063	0.757	0.065
starpilot	cross-encoder test	0.731	0.038	0.783	0.058
fruitbot	IMPALA: train vs. test	0.144	0.012	0.110	0.012
fruitbot	FPN: train vs. test	0.042	0.012	0.085	0.009
fruitbot	cross-encoder train	0.543	0.026	0.518	0.022
fruitbot	cross-encoder test	0.458	0.042	0.533	0.023
heist	IMPALA: train vs. test	–	–	0.104	0.024
heist	FPN: train vs. test	–	–	0.068	0.009
heist	cross-encoder train	–	–	0.593	0.048
heist	cross-encoder test	–	–	0.678	0.041

Table 15: Heist at 25M, the same checkpoints evaluated under both action-selection rules, pooled over encoders (12 measurements per cell, mean $\pm$ SEM). The uniform-random floor on the same level sets (a different level draw) is 3.54 on training levels and 3.15 on held-out levels.

Action selection	Train	Test	Gap
Deterministic	3.38 ± 0.23	0.79 ± 0.16	2.58 ± 0.26
Stochastic	7.17 ± 0.22	2.81 ± 0.16	4.36 ± 0.19

Table 16: Dormant neuron ratio at three thresholds τ , 25M checkpoint, mean $\pm$ SD across 3 seeds.

Env	Encoder	$\tau=0.0$	$\tau=0.025$	$\tau=0.1$
starpilot	IMPALA	$4.5 \pm 1.8\%$	$8.2 \pm 1.7\%$	$16.0 \pm 1.4\%$
starpilot	FPN	$5.3 \pm 1.8\%$	$8.4 \pm 1.5\%$	$14.4 \pm 1.3\%$
fruitbot	IMPALA	$1.2 \pm 0.3\%$	$5.2 \pm 0.6\%$	$15.6 \pm 1.0\%$
fruitbot	FPN	$4.4 \pm 0.8\%$	$8.2 \pm 1.1\%$	$15.3 \pm 0.7\%$

G Why Within-Encoder Train-vs-Test CKA Fails

Linear CKA between $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^{n \times q}$ is $\|X^\top Y\|_F^2 / (\|X^\top X\|_F \|Y^\top Y\|_F)$ after column centering. The numerator depends on the row pairing, so row i of X and row i of Y must be the same input. Our cross-encoder comparisons satisfy this: two encoders, one observation batch. The within-encoder train-vs-test comparison does not; it pairs 1024 train-level observations against 1024 different test-level observations. Table 17 reports four controls on the 8M checkpoints, reproducing the within-encoder rows of Table 14 exactly.

The belief is easy to hold. The analysis script that first computed this statistic justified it in a code comment:

```
CKA does not require the two input sets to be paired/identical -- it
compares the geometry (pairwise similarity structure) of two
representation matrices with the same N -- so this is a legitimate,
if different, question [...]
```

The premise is false for linear CKA, whose cross term $\|X^\top Y\|_F^2$ is a sum over paired rows. We quote it because it is the natural reading of CKA as a “geometry” comparison, and the lesson generalizes beyond this study.

The 2×2 control. The self-comparison returns 1.000000, so the implementation is correct. The values sit far above their row-permutation null— $z = 8$ to 97, the per-seed range across the four rows of Table 17 in the originating run (a regeneration with 200 permutations gives 8.6 to 99.6; the difference is noise in the estimated permutation standard deviation)—so they are not sampling noise. But the 2×2 isolates what drives them. The two dumps have no valid row correspondence. At the 8M budget both were collected with the same number of parallel evaluation environments and written step-major, so row r is the same episode timestep on each side—a genuine correspondence, but by episode time rather than level set. At 25M the sub-environment counts differ (200 train, 100 test), so row r is generally a different timestep on each side (only the first 100 rows share one) and no correspondence exists by level or by time. Cell C removes the domain shift entirely—train against train, disjoint halves of the sub-envs at matched timesteps—and returns the same value as cell A, the train-vs-test comparison. Shuffling one side’s rows ($A \rightarrow B$, $C \rightarrow D$) collapses both toward the null, by $4.6\text{--}7.2\times$ on fruitbot (the per-seed range of A/B and C/D; B and D are single permutation draws, so the endpoints vary between runs). On starpilot every cell already sits at the null, because that representation is close to rank one ($\text{PR} \approx 1.5$); the shuffle contrast is informative only where effective rank is appreciable, while $A \approx C$ holds in both regimes. The statistic responds to timestep alignment and is indifferent to the level set, so it cannot support a claim about train-specificity.

The null in closed form. For column-centered X , Y and a uniform random permutation π , $\mathbb{E}_\pi \|X^\top P_\pi Y\|_F^2 = \text{tr}(XX^\top) \text{tr}(YY^\top) / (n - 1)$, so

$$\mathbb{E}[\text{CKA}_{\text{null}}] = \sqrt{\text{PR}_X \text{PR}_Y / (n - 1)}, \quad \text{PR} = \text{tr}(A)^2 / \|A\|_F^2, \quad A = XX^\top,$$

with PR the participation ratio (effective rank). The identity is exact, not asymptotic: the CKA denominator is permutation-invariant, so the expectation passes through it with no Jensen correction. The agreement between the measured and analytic null columns of Table 17 is therefore a check on the sampling, not evidence of an approximation holding. The between-encoder offset therefore tracks effective rank—on fruitbot, $\text{PR} = 22.6$ for IMPALA against 7.4 for FPN—rather than any property of transfer. **What the encoder difference actually was.** Read naively, IMPALA’s within-encoder value on fruitbot at 8M is $3.44\times$ FPN’s, which would make FPN $3.44\times$ less train-specific. In units of each side’s own analytic null—averaging the statistic and the null over the three seeds, then taking $(\text{CKA}_{\text{IMPALA}} / \text{null}_{\text{IMPALA}}) / (\text{CKA}_{\text{FPN}} / \text{null}_{\text{FPN}})$ —the ratio is $1.14\times$ (the permutation null gives $1.15\times$). Seed by seed it is 1.18, 0.95 and 1.48, so the residual does not even hold its direction across seeds. Almost all of the apparent encoder effect was effective rank.

Heist, and sensitivity to collection geometry. The 2×2 pattern holds on the heist 8M and 25M checkpoints ($A \approx C$, both well above B and D), with cell A restricted to the 500 rows that share a timestep in the 200-versus-100 geometry. The size of the collapse depends on how the observations were collected. In the collection behind Table 14’s heist rows, A/B and C/D range over $3.4\text{--}7.0\times$ pooled across both budgets (against the mean of 200 permutations), comparable to fruitbot. Collecting the same 25M checkpoints with 16 parallel evaluation environments over 64 timesteps instead gives within-encoder values of 0.48 (IMPALA) and 0.43 (FPN), the mean of five level draws, against 0.104

and 0.068 in Table 14, with collapse factors of $41\text{--}92\times$. Same checkpoints, different collection geometry: with 16 environments each contributes 64 consecutive timesteps, so shared episode-time structure dominates and effective rank falls ($\text{PR} \approx 5\text{--}10$). Neither number is wrong. That the statistic moves by $4.6\times$ (IMPALA) and $6.3\times$ (FPN) under nothing but a change in collection order is itself evidence that it measures collection structure rather than the level set.

Table 17: Controls on the within-encoder train-vs-test CKA, 8M checkpoints, mean over 3 seeds. “As computed” is the statistic as originally computed and reproduces Table 14’s 8M column exactly; it and the two null columns are at $n=1024$. The 2×2 block is computed at $n=512$ so its four cells are directly comparable with one another, and is therefore on a different scale from “As computed”: A = timestep-aligned train-vs-test (the $n=512$ analogue of the as-computed quantity), C = timestep-aligned train-vs-train (no domain shift), B and D = the same two with one side’s rows shuffled. B and D are single permutation draws, so the collapse factors A/B and C/D vary slightly between runs.

Env	Enc.	$n=1024$			2×2 ($n=512$)				PR
		As computed	Perm.	Analytic	A	C	B	D	
starpilot	IMPALA	0.020	0.001	0.001	0.008	0.013	0.007	0.002	1.4
starpilot	FPN	0.020	0.002	0.002	0.014	0.012	0.005	0.003	1.6
fruitbot	IMPALA	0.144	0.023	0.023	0.259	0.251	0.039	0.037	22.6
fruitbot	FPN	0.042	0.008	0.008	0.081	0.097	0.015	0.015	7.4