DRED: Zero-Shot Transfer in Reinforcement Learning via Data-Regularised Environment Design

Samuel Garcin [1] **James Doran** [2] **Shangmin Guo** [1] **Christopher G. Lucas** [1] **Stefano V. Albrecht** [1]

Abstract

Autonomous agents trained using deep reinforcement learning (RL) often lack the ability to successfully generalise to new environments, even when these environments share characteristics with the ones they have encountered during training. In this work, we investigate how the sampling of individual environment instances, or levels, affects the zero-shot generalisation (ZSG) ability of RL agents. We discover that, for deep actor-critic architectures sharing their base layers, prioritising levels according to their value loss minimises the mutual information between the agent's internal representation and the set of training levels in the generated training data. This provides a novel theoretical justification for the regularisation achieved by certain adaptive sampling strategies. We then turn our attention to unsupervised environment design (UED) methods, which assume control over level generation. We find that existing UED methods can significantly shift the training distribution, which translates to low ZSG performance. To prevent both overfitting and distributional shift, we introduce *data-regularised environment design* (DRED). DRED generates levels using a generative model trained to approximate the ground truth distribution of an initial set of level parameters. Through its grounding, DRED achieves significant improvements in ZSG over adaptive level sampling strategies and UED methods.

1. Introduction

A central challenge facing modern reinforcement learning (RL) is learning policies capable of zero-shot transfer of learned behaviours to a wide range of environment settings.

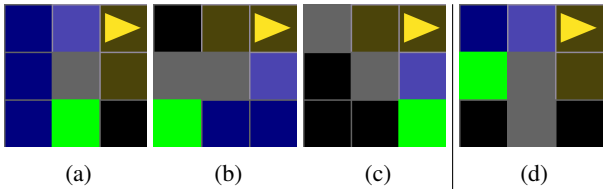


Figure 1: The agent (yellow) must navigate to the goal (green) but cannot pass through walls (grey) and only observes tiles directly adjacent to itself (highlighted yellow). An agent trained over levels (a)-(c) will transfer zero-shot to level (d) if it has learnt a behavior adapted to the task semantics of following blue tiles to the goal location.

Prior applications of RL algorithms (Agostinelli et al., 2019; Lee et al., 2020; Rudin et al., 2022) indicate that strong zero-shot generalisation (ZSG) can be achieved through an adaptive sampling strategy over the set of environment instances available during training, which we refer to as the set of training *levels*. However, the relationship between ZSG and the level sampling process remains poorly understood. In this work, we draw novel connections between this process and the minimisation of an upper bound on the generalisation error. This bound depends on the *mutual information* (MI) between the learned policy π and the set of training levels L,

$$\mathrm{I}(\pi; L) = \mathbf{H}(\mathrm{i}) + \sum_{i \in L} \int_{\bar{\pi} \in \Delta^{\mathbb{A}}} P(\pi{=}\bar{\pi}, \mathrm{i}{=}i) \log P(\mathrm{i}{=}i | \pi{=}\bar{\pi}), \tag{1}$$

where i refers to an individual training level, $\mathbf{H}(\mathrm{i})$ the entropy of the level distribution and $\Delta^{\mathbb{A}}$ the output space of the policy. A model with high $\mathrm{I}(\pi; L)$ outputs action distributions that are highly correlated to the level identity. In other words, the model internally predicts the level it is currently in and learns an ensemble of level-specific policies. In general, such an agent will not transfer to new levels zero-shot, as illustrated in the minimal example in Figure 1. In this scenario, it is possible to identify a training level from its initial observation. An agent with high $\mathrm{I}(\pi; L)$ would use this information to infer the goal location without having to observe it first. In doing so, it can ignore the task semantics shared across levels, while still maximising its returns on the

[1]School of Informatics, University of Edinburgh [2]Huawei. Correspondence to: Samuel Garcin <s.garcin@ed.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

training set. In fact, it can solve level (a) in fewer steps by ignoring these semantics, as there exists a "shortcut" unique to (a). When deployed on (d) after training, the agent will predict it is in (a), since (a) and (d) share the same initial observation. As a result the agent is likely to follow the (a)-specific policy, which will not transfer zero-shot.

As extensively demonstrated in prior work, increasing the number of training levels (Zhang et al., 2018; Cobbe et al., 2020; Packer et al., 2018) or the amount of data generated in each level (e.g. performing data augmentation on observations (Raileanu et al., 2021; Yarats et al., 2021)) reduces the generalisation error, also called *generalisation gap*, defined as the difference in episodic returns when evaluating the agent on the train set and on the full level distribution. From an information theoretic perspective, these approaches induce an implicit information bottleneck on $\mathrm{I}(\pi; L)$ by making any particular level more difficult to identify for a model with a fixed representational capacity. However, it is not clear how certain level sampling strategies achieve smaller generalisation gaps (compared to uniform sampling) without increasing the number of training levels nor augmenting the data generated (Jiang et al., 2021b;a). In Section 3, we show that these strategies may be understood as adaptive rejection sampling mechanisms that prevent data with high $\mathrm{I}(\pi; L)$ from being generated. This allows us to build a connection between these strategies and the minimisation of an upper bound on the generalisation gap that depends on $\mathrm{I}(\pi; L)$. We supplement our findings with an empirical evaluation of different sampling strategies in the Procgen benchmark (Cobbe et al., 2020), and observe a strong correlation between $\mathrm{I}(\pi; L)$ and the generalisation gap.

We then introduce *data-regularised environment design* (DRED) in Section 4. DRED combines adaptive sampling with data augmentation, but does not perform data augmentation across observations but across *levels*. It does so by learning a generative model of the full distribution of levels we would like the agent to transfer to, trained on a limited starting set of levels from that distribution. DRED then employs an adaptive sampling strategy over an augmented set consisting of the starting levels and the generated levels.

While training on an augmented set is effective in preventing overfitting, it may cause distributional shift if that set is not drawn from the same distribution as the original set. We find that existing environment design methods that rely on an unsupervised generation process cause significant distributional shift, and may lead to poor performance at test time. There is therefore a trade-off between augmenting the training set to prevent instance-overfitting (i.e. to prevent learning level-specific policies), and ensuring that this augmented set is consistent with the original distribution (i.e. to avoid distributional shift). We measure DRED's capabilities in a gridworld navigation task that was designed to highlight this trade-off. We find that DRED achieves a smaller generalisation gap than sampling methods restricted to the starting levels, and does so while maintaining a small distributional shift. This differentiates DRED from unsupervised environment design (UED) methods, which do not account for a target task or target distribution. DRED achieves significant improvements in the agent's ZSG capabilities, reaching 1.2 times the returns of the next best baseline on held-out levels. It also improves performance by two to three times on more difficult instantiations of the target task.[1]

2. Preliminaries

Reinforcement learning. We model an individual level as a Partially Observable Markov Decision Process (POMDP) $\langle \mathbb{A}, \mathbb{O}, \mathbb{S}, \mathcal{T}, \Omega, R, p_0, \gamma \rangle$ where $\mathbb{A}$ is the action space, $\mathbb{O}$ is the observation space, $\mathbb{S}$ is the set of states, $\mathcal{T} : \mathbb{S} \times \mathbb{A} \to \Delta^{\mathbb{S}}$ and $\Omega : \mathbb{S} \to \Delta^{\mathbb{O}}$ are the transition and observation kernels (denoting $\Delta^{\mathbb{S}}$ as the set of all possible probability distributions over $\mathbb{S}$), $R : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the reward function, $p_0(\mathrm{s})$ is the initial state distribution and γ is the discount factor. We consider the episodic RL setting, in which the agent attempts to learn a policy π maximising the expected discounted return $V^{\pi}(s_t) = \mathbb{E}_{\pi}[\sum_{\tilde{t}=t}^{T} \gamma^{t-\tilde{t}} r_t]$ over an episode terminating at timestep T, where s_t and r_t are the state and reward at step t. We use V^{π} to refer to $V^{\pi}(s_0)$, the expected episodic returns taken from the first timestep of the episode. In this work, we focus on on-policy actor-critic algorithms (Mnih et al., 2016; Lillicrap et al., 2016; Schulman et al., 2017) representing the agent policy $\pi_{\boldsymbol{\theta}}$ and value estimate $\hat{V}_{\boldsymbol{\theta}}$ with neural networks (we use $\boldsymbol{\theta}$ to refer to model weights). The policy and value networks usually share an intermediate state representation $b_{\boldsymbol{\theta}}(o_t)$ (or for recurrent architectures $b_{\boldsymbol{\theta}}(H_t^o)$, $H_t^o = \{o_0, \cdots, o_t\}$ being the history of observations o_i).

Contextual MDPs. Following Kirk et al. (2023), we model the set of environment instances we aim to generalise over as a Contextual-MDP (CMDP) $\mathcal{M} = \langle \mathbb{A}, \mathbb{O}, \mathbb{S}, \mathcal{T}, \Omega, R, p_0(\mathrm{s}|\mathbf{x}), \gamma, \mathbb{X}_C, p(\mathbf{x}) \rangle$. In a CMDP, the reward function and the transition and observation kernels also depend on the context set $\mathbb{X}_C$ with associated distribution $p(\mathbf{x})$, that is we have $\mathcal{T} : \mathbb{S} \times \mathbb{X}_C \times \mathbb{A} \to \Delta^{\mathbb{S}}$, $\Omega : \mathbb{S} \times \mathbb{X}_C \to \Delta^{\mathbb{O}}$, $R : \mathbb{S} \times \mathbb{X}_C \times \mathbb{A} \to \mathbb{R}$. Each element $\boldsymbol{x} \in \mathbb{X}_C$ is not observable by the agent and instantiates a level $i_{\boldsymbol{x}}$ of the CMDP with initial state distribution $p_0(\mathrm{s}|\mathbf{x})$. The CMDP is equivalent to a POMDP if we consider its state space to be $\mathbb{S} \times \mathbb{X}_C$, which means that the agent is always in a partially observable setting, even when the state space of individual levels is fully observable.

We assume access to a parametrisable simulator with pa-

[1] Our code and experimental data are available at https://github.com/uoe-agents/dred.

rameter space $\mathbb{X}$, with $\mathbb{X}_C \subset \mathbb{X}$. While prior work expects $\mathbb{X}_C$ to correspond to all solvable levels in $\mathbb{X}$ (often defined as the levels in which a positive return is achievable), we will extend our analysis to the more general setting in which there may be more than one CMDP within $\mathbb{X}$, whereas we aim to solve a specific target CMDP. We refer to levels with parameters $\boldsymbol{x} \in \mathbb{X}_C$ as *in-context* and to levels outside of this set as *out-of-context*.

The generalisation gap. We start training with access to a limited set of level parameters $X_{\text{train}} \subset \mathbb{X}_C$ sampled from $p(\mathbf{x})$, and evaluate generalisation using a set of held-out level parameters X_{test}, also sampled from $p(\mathbf{x})$. We can estimate generalisation error using a formulation reminiscent of supervised learning,

$$\text{GenGap}(\pi) \coloneqq \frac{1}{|X_{\text{train}}|} \sum_{\boldsymbol{x} \in X_{\text{train}}} V^{\pi}_{i_{\boldsymbol{x}}} - \frac{1}{|X_{\text{test}}|} \sum_{\boldsymbol{x} \in X_{\text{test}}} V^{\pi}_{i_{\boldsymbol{x}}}. \tag{2}$$

Bertrán et al. (2020) extend generalisation results in the supervised setting (Xu & Raginsky, 2017) to obtain an upper bound for the GenGap.

Theorem 2.1. *For any CMDP such that* $|V^{\pi}_{C}(H^{o}_{t})| \leq D/2, \forall H^{o}_{t}, \pi$*, with* D *being a constant, then for any set of training levels* L*, and policy* π

$$\textit{GenGap}(\pi) \leq \sqrt{\frac{2D^2}{|L|} \times I(L;\pi)}. \tag{3}$$

We will show that minimising this bound is an effective surrogate objective for reducing the GenGap.

Adaptive level sampling. We study the connection between $\mathrm{I}(L;\pi)$ and adaptive sampling strategies over L. Prioritised Level Replay (PLR, Jiang et al., 2021b) introduce a scoring function $\textbf{score}(\tau_i, \pi)$ which compute level scores from trajectory rollouts τ_i. Scores are used to define an adaptive sampling distribution over a level buffer Λ, with

$$P_{\Lambda} = (1-\rho) \cdot P_S + \rho \cdot P_R, \tag{4}$$

where P_S is a distribution parametrised by the level scores and ρ is a coefficient mixing P_S with a staleness distribution P_R that promotes levels replayed less recently. Jiang et al. (2021b) experiment with different scoring functions, and empirically find that the scoring function based on the ℓ_1-value loss $S^{V}_{i} = \textbf{score}(\tau_i, \pi) = (1/|\tau_i|) \sum_{H^{o}_{t} \in \tau_i} |\hat{V}(H^{o}_{t}) - V^{\pi}_{i}(H^{o}_{t})|$ incurs a significant reduction in the GenGap at test time.

In the remaining sections, we draw novel connections between the ℓ_1-value loss prioritisation strategy and the minimisation of $\mathrm{I}(L;\pi)$. We then introduce DRED, a level generation and sampling framework training the agent over an augmented set of levels. DRED jointly minimises $\mathrm{I}(L;\pi)$ while increasing $|L|$ and as such is more effective at minimising the bound from Theorem 2.1.

3. How does adaptive level sampling impact generalisation in RL?

In this section, we establish that adaptive sampling strategies reduce the GenGap by inducing an *information bottleneck* on $\mathrm{I}(L;\pi)$. We begin our analysis by deriving a method for empirically estimating an upper bound for $\mathrm{I}(L;\pi)$. We find this upper bound and the generalisation gap to be well correlated for different sampling strategies in the Procgen benchmark (Cobbe et al., 2020), a benchmark of 16 games designed to measure generalisation in RL. As in prior work (Jiang et al., 2021b), we observe that ℓ_1-value loss prioritisation reduces the GenGap while improving sample efficiency. Here we take a further step, and aim to understand *why* value loss prioritisation minimises the GenGap.[2] We do so by establishing a connection between the value loss and $\mathrm{I}(L;\pi)$. We conclude this section by highlighting the importance of maintaining a low distributional shift during training, an important feature of DRED, which we will introduce in the next section.

3.1. Mutual information estimation

$\mathrm{I}(L;\pi)$ is often difficult to measure from the model outputs alone. We estimate instead $\mathrm{I}(L;b)$, the mutual information at the last shared layer between the actor and critic. As shown in the following lemma, $\mathrm{I}(L;b)$ upper bounds $\mathrm{I}(L;\pi)$.

Lemma 3.1. *(proof in appendix) Given a set of training levels* L *and policy* $\pi = f \circ b$*, where* $b(H^{o}_{t}) = h_t \in \mathbb{B}$ *is an intermediate representation function and* $f : \mathbb{B} \to \Delta^{\mathbb{A}}$ *maps to the agent's action distribution, we have*

$$I(L;\pi) \leq \mathbf{H}(\mathrm{i}) + \sum_{i \in L} \int_{\mathbb{B}} p(h,i) \log p(i|h) dh, \tag{5}$$

where the right-hand side is equivalent to $I(L;b)$*.*

This result applies to any state representation function b, including the non-recurrent case where $b(H^{o}_{t}) = b(o_t)$. To remain consistent with the CMDP, we must set $p(\mathrm{i})$ to $p(\mathbf{x})$, making the entropy $\mathbf{H}(\mathrm{i})$ a constant. However, the second term in Equation (5) depends on the output of the learned representation $b_{\boldsymbol{\theta}}$, and may be empirically estimated as

$$\sum_{i \in L} \int_{\mathbb{B}} p(h,i) \log p(i|h) dh \approx \frac{1}{N} \sum_{n}^{N} \log p_{\boldsymbol{\theta}}(i^{(n)} | \boldsymbol{h}^{(n)}), \tag{6}$$

where $p_{\boldsymbol{\theta}}$ predicts the current level from representations $\boldsymbol{h}_t = b_{\boldsymbol{\theta}}(H^{o}_{t})$ collected during rollouts.

[2] For a discussion on the relationship of value loss and sample efficiency, we refer the reader to Schaul et al. (2016).

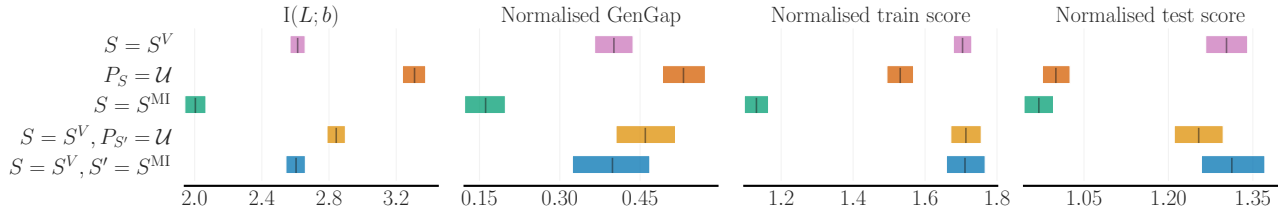


Figure 2: Aggregated $\mathrm{I}(L;b)$, GenGap, train and test scores of different sampling strategies over 5 seeds across all Procgen games, using the rliable library (Agarwal et al., 2021). Coloured boxes indicate the 95% confidence interval. For each game, the train and test scores and the GenGap are normalised by the mean score of $(P_S = \mathcal{U})$ over the test set. Per-game scores, mutual information and classifier accuracy are reported in Appendix B, and Appendix E.1 provides an extended description of our experimental setup.

3.2. Comparing adaptive sampling strategies

To evaluate their impact on the GenGap, we compare different adaptive sampling strategies in the "easy" setting of Procgen ($|L| = 200$, $25M$ timesteps). We use the default PPO (Schulman et al., 2017) architecture and hyperparameters proposed by Cobbe et al. (2020), which uses a non-recurrent intermediate representation $b_{\boldsymbol{\theta}}(o_t)$. We measure $\mathrm{I}(L;b)$ using Equation (6), parametrising $p_{\boldsymbol{\theta}}$ as a linear classifier. Figure 2 compares uniform sampling $(P_S = \mathcal{U})$ with adaptive sampling strategies obtained from different level scoring functions in Equation (7). We compare value loss scoring $(S = S^V)$ with $(S = S^{\mathrm{MI}})$, a scoring function that directly prioritises levels with low $\mathrm{I}(L;b)$. $(S = S^{\mathrm{MI}})$ exploits that Equation (6) may be decomposed into level specific terms $I_i = \sum_t^T \log p_{\boldsymbol{\theta}}(i|b_{\boldsymbol{\theta}}(o_t))$ and uses scoring strategy $\textbf{score}(\tau_i, \pi) = I_i$.

We also consider mixed strategies $(S = S^V, S' = S^{\mathrm{MI}})$ and $(S = S^V, P_{S'} = \mathcal{U})$, which are obtained by introducing a secondary scoring function to Equation (4),

$$P_\Lambda = (1-\rho)\cdot((1-\eta)\cdot P_S + \eta \cdot P_{S'}) + \rho \cdot P_R, \quad (7)$$

where η is the mixing coefficient, which may be scheduled over the course of training. We report our main observations below, with extended analysis and results available in Appendix B.

First observation. Minimising $\mathrm{I}(L;b)$ on the generated training data using an adaptive distribution results in agent representations with reduced $\mathrm{I}(L;b)$ under the *original distribution*.

Second observation. We find this representation to be highly informative of the level identity. Out of 200 training levels, our linear classifier predicts the current level with 49% accuracy with $(P_S = \mathcal{U})$ and its accuracy only drops to 35% with $(S = S^V)$ and 19% with $(S = S^{\mathrm{MI}})$. This indicates that observations get mapped to level-specific *clusters* in the representation space, which let the agent learn level-specific policies.

Third observation. We measure a strong positive correlation ($\rho = 0.6$) and rank correlation (Kendall $\xi = 0.5$, $p < 1e-50$) between $\mathrm{I}(L;b)$ and the GenGap, across all procgen games and sampling strategies tested. This makes $\mathrm{I}(L;b)$ a useful proxy for the GenGap. $\mathrm{I}(L;b)$ has lower variance, does not require normalisation across environments and, crucially, *does not necessitate a held-out test set of levels* to be measured.

Fourth observation. $(S = S^{\mathrm{MI}})$ achieves the highest reductions in $\mathrm{I}(L;b)$ and the GenGap. However, it also significantly degrades performance during training, and is the worst performing strategy when considering final test scores. This result is consistent with Theorem 2.1 and Lemma 3.1, as $\mathrm{I}(L;b)$ bounds the GenGap and not the test returns.[3] $(S = S^V)$ strikes a good balance between improving sample efficiency and reducing the GenGap. Indeed, our best performing mixing schedule for $(S = S^V, S' = S^{\mathrm{MI}})$ (reported here) only achieved a very marginal (and not statistically significant) improvement in test score and GenGap over $(S = S^V)$.

Fifth observation. Under $(S = S^V, P_{S'} = \mathcal{U})$ training scores are similar to $(S = S^V)$ but both the GenGap and $\mathrm{I}(L;b)$ increase. We hypothesise that the mechanisms responsible for improving sample efficiency and generalisation in $(S = S^V)$ are different, and that the latter is achieved through adaptive $\mathrm{I}(L;b)$ regularisation.

3.3. Value loss prioritisation reduces $\mathrm{I}(L;b)$

While it is intuitive that an adaptive strategy such as $(S = S^{\mathrm{MI}})$ minimises $\mathrm{I}(L;b)$ by preventing the agent to train on informative levels, the relationship between $(S = S^V)$ and $\mathrm{I}(L;b)$ is less evident. To understand this relationship it is helpful to remember that $(S = S^V)$ *rejects* levels with low

[3] Excessive data regularisation is not desirable: in the most extreme case, destroying all information contained within the training data would guarantee $\mathrm{I}(L;b) = \text{GenGap} = 0$ but it would also make the performance on the train and test sets equally bad.

value loss as much as it prioritises levels with high value loss; and to consider what achieving low value loss means in the CMDP setting.

We can estimate the value function V_C^π of a CMDP from level specific value functions V_i^π by employing the unbiased value estimator lemma from Bertrán et al. (2020).

Lemma 3.2. *Given a policy π and a set of levels L from a CMDP, we have $\forall H_t^o \ (t < \infty)$ compatible with L (meaning that the observation sequence H_t^o occurs in L), $\mathbb{E}_{L|H_t^o}[V_i^\pi(H_t^o)] = V_C^\pi(H_t^o)$, with $V_i^\pi(H_t^o)$ being the expected returns under π given observation history H_t^o in a given level i, and $V_C^\pi(H_t^o)$ being the expected returns across all possible occurrences of H_t^o in the CMDP.*

It follows that the value prediction loss also employs level-specific targets, with

$$L_V(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n}^{N}(\hat{V}_{\boldsymbol{\theta}}(H_t^{o(n)}) - V_i^{\pi(n)})^2, \tag{8}$$

and Lemma 3.2 guaranteeing convergence to an unbiased estimator for V_C^π when minimising Equation (8).

Equivalently, we can express each level-specific value function V_i^π as a combination of the CMDP value function V_C^π and a level-specific function v_i^π, Lemma 3.2 ensuring that the v_i^π functions cancel out in expectation. Nevertheless, when v_i^π is not zero everywhere, identifying the current level i and predicting v_i^π are both necessary to predict individual value targets V_i^π. Perfect value prediction over the training levels (i.e. overfitting and reaching zero value loss) therefore necessitates learning an intermediate representation from which the current level i is identifiable, implying high $\mathrm{I}(L; b)$. It follows that, by rejecting levels with low value loss, value loss prioritisation prevents the agent from collecting data from the levels in which b has begun to overfit.

In other words, value loss prioritisation acts as a form of *rejection sampling* that prevents data with high $\mathrm{I}(L; b)$ from being generated. This rejection sampling is adaptive, as $L_V(\boldsymbol{\theta})$ depends on the current learned weights $\boldsymbol{\theta}$, and thus continually resists convergence to a representation overfitting to any particular level.

3.4. Distributional shift and the effect of increasing $|L|$

In 3 out of 16 Procgen games, $(S = S^V)$ does not improve over $(P_S = \mathcal{U})$ on the test set, and does worse on the train set (per-game results are available in Appendix B.3). As we are interested in combining adaptive sampling with the generation of additional levels, we conduct a preliminary experiment investigating how the inclusion of additional training levels impacts ZSG in "Miner", one of the games in which $(S = S^V)$ underperforms.

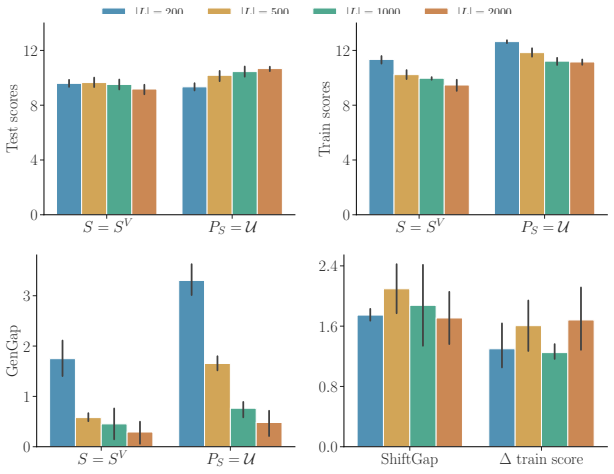


Figure 3: Investigating the effect of increasing $|L|$ when adaptive sampling induces distributional shift. Top row reports the test and train performance; the GenGap is on the bottom left; on the bottom right is compared, for $(S = S^V)$, the ShiftGap and the score differential with $(P_S = \mathcal{U})$ over the train set. Vertical bars indicate standard error over 5 seeds.

In Figure 3 we observe larger L results in a reduction in the GenGap and in an increase in test scores under uniform sampling, as reported in several prior studies (Zhang et al., 2018; Cobbe et al., 2020; Packer et al., 2018). For $(S = S^V)$ we also observe a reduction in GenGap but it is caused by scores dropping over the train set instead of improving over the test set. This phenomenon is not optimisation related, as train scores for $(S = S^V)$ under its own adaptive distribution remain in-line with the $(P_S = \mathcal{U})$ train scores. In fact it is caused by distributional shift, a potential pitfall of adaptive distributions. In Miner, the shifted training distribution prevents the agent to improve its test scores further, even as the number of training levels is increased.

As this phenomenon is not captured by the GenGap, we propose the *shift-induced gap*, or ShiftGap, a complementary metric quantifying the performance reduction induced by distributional shift under non-uniform training distributions,

$$\text{ShiftGap}(\pi) \coloneqq \sum_{i\in\Lambda} P_\Lambda(i)\cdot V_i^\pi - \frac{1}{|L|}\sum_{i\in L} V_i^\pi. \tag{9}$$

Unlike the GenGap, the ShiftGap does not necessitates held-out test levels to be measured, and only relies on scores being normalised across levels.[4] We find that the ShiftGap for $(S = S^V)$ closely matches (within measurement error) the performance differential between $(S = S^V)$ and $(P_S = \mathcal{U})$ over the train set. In later experiments, we will show

[4]Note that using X_{test} in the second term makes ShiftGap $\equiv$ GenGap when $P_\Lambda = \mathcal{U}$, whereas it should be 0.

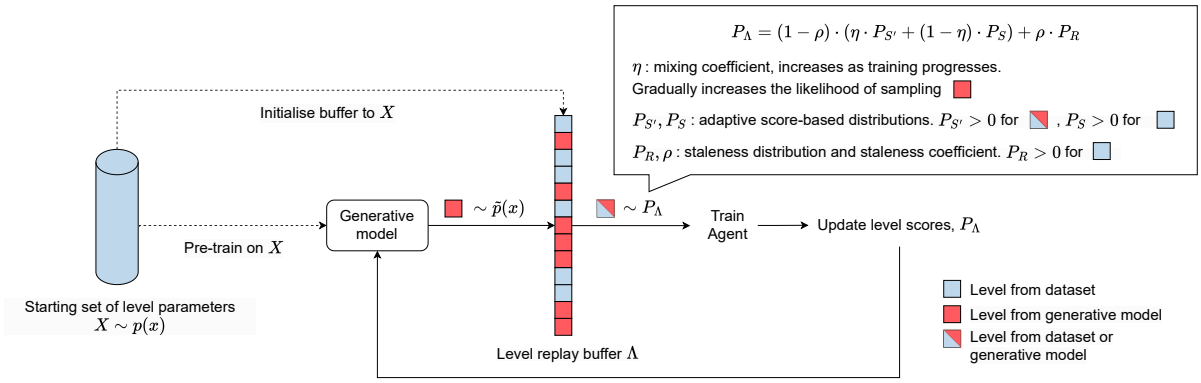


Figure 4: Overview of the data-regularised environment design framework.

it is important for the ShiftGap to remain small in order to achieve strong ZSG.

Algorithm 1 Data-regularised environment design

Input: Pre-trained encoder and decoder networks $\psi_{\boldsymbol{\theta}_E}$, $\phi_{\boldsymbol{\theta}_D}$, level parameters X_{train}, number of pairs M, number of interpolations per pair K

Initialise level buffer Λ to level parameters in X_{train}
Initialise agent policy π
while *not converged* **do**
  Instantiate levels from batch $X \sim P_\Lambda$ and collect rollouts, update level scores S, S' in Λ
  Update π using collected rollouts
  Uniformly sample M pairs from X_{train}
  for $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in pairs **do**
    Compute latent parameters using $\psi_{\boldsymbol{\theta}_E}$ and K interpolations between $(\boldsymbol{\mu}_\mathbf{z}, \boldsymbol{\sigma}_\mathbf{z})_i$ and $(\boldsymbol{\mu}_\mathbf{z}, \boldsymbol{\sigma}_\mathbf{z})_j$
    for $(\boldsymbol{\mu}_\mathbf{z}, \boldsymbol{\sigma}_\mathbf{z})_k$ in interpolations **do**
      Sample embedding $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{z}, \boldsymbol{\sigma}_\mathbf{z})$
      Instantiate $\tilde{\boldsymbol{x}} \leftarrow \phi_{\boldsymbol{\theta}_D}(\boldsymbol{z})$ and compute s, s'
      Add $\langle \tilde{\boldsymbol{x}}, s, s' \rangle$ to Λ

4. Data-regularised environment design

We have established that we can reduce $\mathrm{I}(L; b)$ and the GenGap by increasing $|L|$ or by employing an adaptive sampling strategy. However, we have observed that increasing $|L|$ may not result in higher test set performance when there is significant distributional shift during training. As the CMDP context distribution $p(\mathbf{x})$ is rarely known in practical applications, artificially generating extra training levels is an additional source of distributional shift. To capitalise on the benefits provided by adaptive sampling and level generation, while limiting distributional shift, we propose *data-regularised environment design* (DRED), a framework that combines adaptive sampling with a level generation process approximating $p(\mathbf{x})$.

Instead of direct knowledge of $p(\mathbf{x})$, we assume having access to a limited set of level parameters $X_{\text{train}} \sim p(\mathbf{x})$. Each $\boldsymbol{x} \in X_{\text{train}}$ instantiates a level $i_{\boldsymbol{x}}$ from the CMDP. We are allowed to sample from the full simulator parameter space $\mathbb{X}$, which means we can *augment* our set of training levels with new levels $\tilde{\mathbf{x}} \in \mathbb{X}$.

DRED consists of two components: a *generative phase*, in which an augmented set $\tilde{X}$ is generated from a batch $X \sim \mathcal{U}(X_{\text{train}})$ and is added to the buffer Λ, and a *replay phase*, in which we use the adaptive distribution P_Λ to sample levels from the buffer. We alternate between the generative and replay phases, and only perform gradient updates on the agent during the replay phase. Algorithm 1 and Figure 4 describe the full DRED pipeline, and we provide further details on each phase below.

4.1. The generative phase

We initialise the buffer Λ to contain X_{train} levels and gradually add generated levels $\tilde{X}$ over the course of training. DRED is not restricted to a particular approach to obtain $\tilde{X}$. In this work, we use a VAE (Kingma & Welling, 2014; Rezende et al., 2014) and refer to our method as VAE-DRED. The VAE models the underlying training data distribution $p(\mathbf{x})$ as stochastic realisations of a latent distribution $p(\mathbf{z})$ via a generative model $p(\mathbf{x} \mid \mathbf{z})$. The model is pre-trained on X_{train} by maximising the variational ELBO

$$\mathcal{L}_{\text{ELBO}} = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p(\mathbf{x})}\Big[\mathop{\mathbb{E}}_{\boldsymbol{z}\sim q(\mathbf{z}|\mathbf{x};\psi_{\boldsymbol{\theta}_E})}[\log p(\mathbf{x} \mid \mathbf{z}; \phi_{\boldsymbol{\theta}_D})] - \beta D_{\text{KL}}(q(\mathbf{z} \mid \mathbf{x}; \psi_{\boldsymbol{\theta}_E}) \,||\, p(\mathbf{z}))\Big], \tag{10}$$

where $q(\mathbf{z} \mid \mathbf{x}; \psi_{\boldsymbol{\theta}_E})$ is a variational approximation of an intractable model posterior distribution $p(\mathbf{z} \mid \mathbf{x})$ and $D_{\text{KL}}(\cdot \,||\, \cdot)$ denotes the Kullback–Leibler divergence, which is balanced using the coefficient β, as proposed by Higgins et al. (2017). The generative $p(\mathbf{x} \mid \mathbf{z}; \phi_{\boldsymbol{\theta}_D})$ and variational $q(\mathbf{z} \mid \mathbf{x}; \psi_{\boldsymbol{\theta}_E})$ models are parametrised via encoder and decoder networks $\psi_{\boldsymbol{\theta}_E}$ and $\phi_{\boldsymbol{\theta}_D}$.

Maximising Equation (10) fits the VAE such that $p(\mathbf{x}; \phi_{\boldsymbol{\theta}_D}) = \int p(\mathbf{x} \mid \mathbf{z}; \phi_{\boldsymbol{\theta}_D}) p(\mathbf{z}) \, \mathrm{d}\mathbf{z} \approx p(\mathbf{x})$. Out-of-context levels are less likely with $\tilde{\boldsymbol{x}} \sim p(\mathbf{x}; \phi_{\boldsymbol{\theta}_D})$ than with $\tilde{\boldsymbol{x}} \sim \mathcal{U}(\mathbb{X})$, and we show in Section 5 that this aspect is key in enabling DRED agents to outperform UED agents. We follow White (2016) and interpolate $\mathbf{z}$ in the latent space between the latent representations of a pair of samples $(\boldsymbol{x}_i, \boldsymbol{x}_j) \sim X_{\text{train}}$ (instead of sampling $\mathbf{z}$ from $p(\mathbf{z})$), as this improves the quality of the generated $\tilde{\boldsymbol{x}}$.

After generating a batch of levels parameters $\tilde{X}$ we collect rollouts (without updating agent weights) to compute their scores, adding to the buffer Λ any level scoring higher than the lowest scoring generated level in Λ. We only consider levels solved at least once during rollouts for inclusion, to ensure that unsolvable levels do not get added in. We provide additional details on the VAE architecture, hyperparameters and pre-training process in Appendix E.3.

4.2. The replay phase

All levels in X_{train} originate from $p(\mathbf{x})$ and are in-context, whereas generated levels, which are obtained from an approximation of $p(\mathbf{x})$, do not benefit from as strong of a guarantee. As training on out-of-context levels can significantly harm the agents' performance on the CMDP, we control the ratio between X_{train} and augmented levels using the mixed scoring introduced in Equation (7) for P_Λ. The support of P_S and P_R is limited to X_{train}, whereas $P_{S'}$ supports the entire buffer. We set both S and S' to score levels according to the ℓ_1-value loss. We find out-of-context levels to be particularly harmful in the early stages of training, and limit their occurrence early on by linearly increasing η from 0 to 1 over the course of training.

5. DRED experiments

Our experiments seek to answer the following questions: 1) How important is it to remain grounded to the target CMDP when generating additional levels, instead of simply maximising level diversity? 2) Is DRED successful in grounding the training distribution to the target CMDP, and does it improve transfer to held-out levels and edge-cases?

5.1. Experimental setup

We choose Minigrid (Chevalier-Boisvert et al., 2018), a partially observable gridworld navigation domain, for our experiments. Despite its simplicity, Minigrid has a controllable level parameter space (unlike Procgen), and levels are parameterised to vectors describing the locations, starting states and appearance of the objects in the grid.

When benchmarking UED methods (and RL algorithms in general) it is implicitly agreed upon that each (solvable) level instantiated belongs to the target CMDP. Yet, this is rarely true in a practical application, as in-context level parameters $\mathbb{X}_C$ often correspond to a small and highly structured region of the simulator parameter space $\mathbb{X}$. With this is mind, we seek to design a target CMDP with similar properties in Minigrid. We define the context space of our target CMDP as spanning the layouts where the location of green "moss" tiles and orange "lava" tiles are respectively positively and negatively correlated to their distance to the goal location. We employ procedural generation to obtain a set X_{train} of 512 level parameters (we refer the reader to Figure 17 for a visualisation of levels from X_{train}, and to Appendix D for extended details on the CMDP specification and procedural generation processes).

As the agent only observes its immediate surroundings and does not know the goal location a priori, the optimal CMDP policy is one that exploits the semantics shared by all levels in the CMDP, exploring first areas with high perceived moss density and avoiding areas with high lava density. Other CMDPs exist in the level space, and may correspond to incompatible optimal policies (for example a CMDP in which the correlation of moss and lava tiles with the goal is reversed). As such, it is important to maintain consistency with the CMDP semantics when generating new levels.

Our first set of baselines is restricted to sample from X_{train}, and consists of uniform sampling ($\mathcal{U}$) and sampling using the ℓ_1-value loss strategy (PLR). Our second set uses level generation, removing this restriction. We consider domain randomisation (DR) (Tobin et al., 2017) which generates levels by sampling uniformly between pre-determined ranges of parameters; RPLR (Jiang et al., 2021a), which combines PLR with DR used as its generator; and the current UED state-of-the-art, ACCEL (Parker-Holder et al., 2022), an extension of RPLR replacing DR by a generator making local edits to currently high scoring levels in the buffer. In all experiments we train the same PPO (Schulman et al., 2017) agent for 27k updates. Additional details on our implementation are provided in Appendix E.2.

5.2. Results

ZSG to held-out levels. As shown in Figure 5 (top), VAE-DRED achieves statistically significant improvements in its IQM (inter-quantile mean), mean score, optimality gap (compared to the level-specific optimal policy) and mean solved rate over other methods on held-out levels from the CMDP. VAE-DRED drastically increases the number of training levels available to the agent while remaining consistent with the target CMDP. VAE-DRED maintains a small distributional shift, which we quantify in our extended analysis in Appendix C.1, and a low ShiftGap throughout training (Figure 13). This is thanks to its generative model effectively approximating $p(\mathbf{x})$, and to its mixed sampling strategy ensuring levels from X_{train} are sampled often, and even

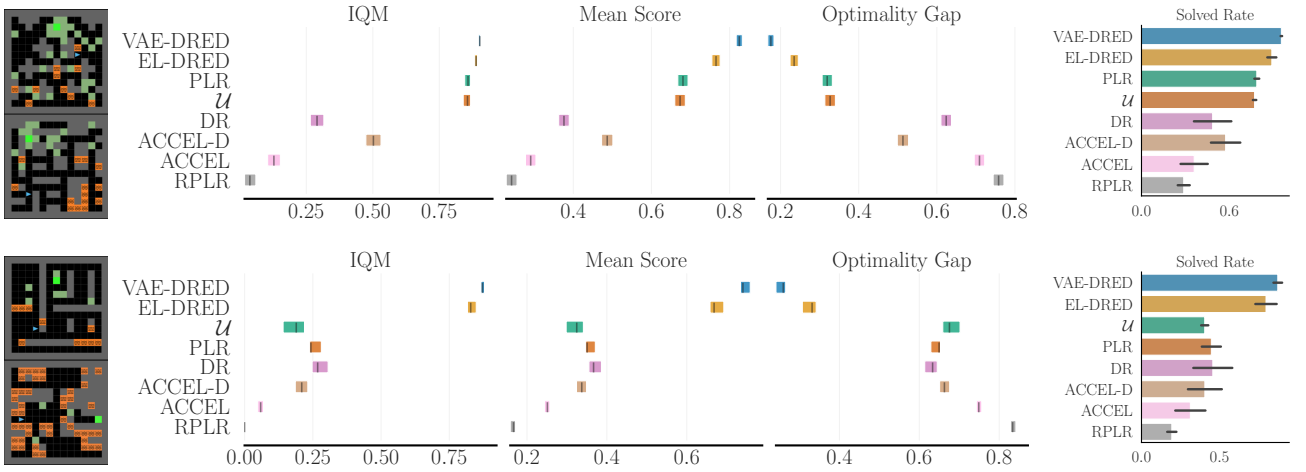


Figure 5: Aggregate final performance and mean solved rate on X_{test}, an evaluation set of 2048 levels sampled from $p(\mathbf{x})$ and held-out during training (top), and on 448 in-context edge cases (bottom). Example layouts from each evaluation set are plotted on the left. The coloured boxes indicate a 99% confidence interval and the black horizontal bars indicate standard error across 5 training seeds. We refer the reader to Appendix D for additional details on our evaluation sets.

more-so early-on. Despite small GenGap (Figure 13), UED baselines achieve low test scores, as they perform poorly on both the test set and on X_{train}. Further analysis conducted in Appendix C.1 confirms that, in general, UED methods incur larger distributional shift than DRED or adaptive sampling strategies, and result in larger ShiftGap.

ZSG to edge cases. We next investigate whether VAE-DRED's level generation improves robustness to in-context *edge cases* with a near zero likelihood of occurring in X_{train} (Figure 5, bottom). We find VAE-DRED to be particularly dominant in this setting, achieving over three times DR's IQM, the next best method, and twice its solved rate and mean score. VAE-DRED makes the agent robust to edge cases by introducing additional diversity in the training levels. These generated levels remain consistent with the CMDP semantics, as can be observed qualitatively in renderings of generated levels in Figure 19 and quantitatively in the training distribution metrics reported in in Figure 14.

ZSG to hard levels. In Figures 6 and 16, we evaluate transfer to in-context "Hardcore" levels. Being 9 times larger in area than training levels, Hardcore levels are significantly more challenging to solve, even for Humans. This setting is where the performance gap between VAE-DRED and other methods is the largest, with VAE-DRED solving three times as many levels as the next best baseline.

Ablations. To better understand the importance of the pre-trained generative model, we introduce EL-DRED, which replaces the VAE with ACCEL's local edit strategy. EL-DRED may be viewed as a DRED variant of ACCEL augmenting X_{train} using a non-parametric generative method, or, equivalently, as an ablation of VAE-DRED that does not approximate $p(\mathbf{x})$, and is therefore less grounded to the CMDP. EL-DRED outperforms all other methods in each of the level sets depicted above, with the exception of VAE-DRED. In Figure 6 (bottom right), we compare the two methods, and find that VAE-DRED remains significantly more likely to outperform EL-DRED in each level set. Finally, ACCEL-D shows that initialising the buffer to X_{train} isn't sufficient for preventing ACCEL's editing mechanism to rapidly incur significant distributional shift. The only difference between EL-DRED and ACCEL-D is η being set to 1 throughout training (see the text box in Figure 4 for a depiction of the role of η in DRED). Yet, the gap in performance is significant, and highlights the importance of avoiding out-of-context levels early in training.

6. Related work

Buffer-free sampling strategies. Domain randomisation (DR, Tobin et al., 2017; Jakobi, 1997), is one of the earliest proposed methods for improving the generalisation ability of RL agents by augmenting the set of available training levels. It does so by sampling uniformly between manually specified ranges of environment parameters. Subsequent contributions introduce an implicit prioritisation over the generated set by inducing a minimax return (robust adversarial RL, Pinto et al., 2017) or a minimax regret game (UED, Dennis et al., 2020) between the agent and a *level generator*, which are trained concurrently. These adversarial formulations prioritise levels in which the agent is currently performing poorly to encourage robust generalisation over the sampled set, with UED achieving better Nash

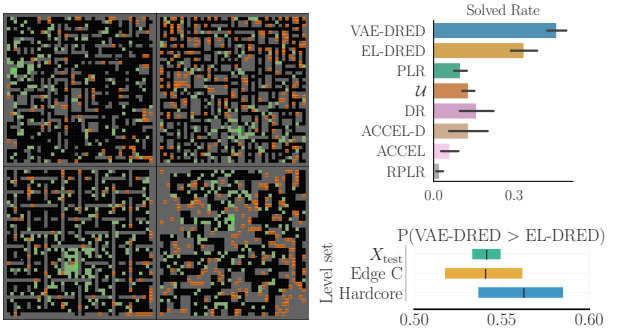


Figure 6: Top right: Solved rate achieved on 216 "Hardcore" levels (examples plotted on the left). Bottom right: Probability of VAE-DRED achieving higher zero-shot returns than its ablation EL-DRED, on different evaluation sets. Coloured boxes indicate the 99% confidence interval.

equilibrium theoretical guarantees. CLUTR (Azad et al., 2023) removes the need for domain-specific RL environments and improves sample efficiency by having the level generator operate within a low dimensional latent space of a generative model pre-trained on randomly sampled level parameters. However, buffer-free methods remain vastly outperformed by a well calibrated DR implementation and the buffer-based sampling strategies discussed next.

Buffer-based sampling strategies. Prioritised sampling is often applied to off-policy algorithms, where individual interactions within the replay buffer are prioritised (Schaul et al., 2016) or resampled with different goals in multi-goal RL (Andrychowicz et al., 2017; Zhang et al., 2020). Prioritised Level Replay (PLR, Jiang et al., 2021b) instead affects the sampling process of *future* experiences, and is thus applicable to both on- and off-policy algorithms. PLR maintains a buffer of training levels and empirically demonstrates that prioritising levels using a scoring function proportional to high value prediction loss results in higher sample efficiency and improved ZSG performance. Robust PLR (RPLR, Jiang et al., 2021a) extends PLR to the UED setting, using DR as its level generation mechanism, while ACCEL (Parker-Holder et al., 2022) gradually evolves new levels by performing random edits on high scoring levels in the buffer. SAMPLR (Jiang et al., 2022) proposes to eliminate the distributional shift induced by the prioritisation strategy by modifying individual interactions using a second simulator that runs in parallel. However, SAMPLR only applies to settings with direct access to the ground truth context distribution, while DRED learns to approximate this distribution.

Mutual information minimisation in RL. In prior work, mutual information has been minimised in order to mitigate instance-overfitting, either by learning an ensemble of policies (Bertrán et al., 2020; Ghosh et al., 2021), performing data augmentation on observations (Raileanu et al., 2021; Yarats et al., 2021), an auxiliary objective (Dunion et al., 2024; Dunion & Albrecht, 2024) or introducing information bottlenecks through selective noise injection on the agent model (Igl et al., 2019; Cobbe et al., 2019). In contrast, our work is the first to draw connections between mutual information minimisation and adaptive level sampling.

7. Conclusion

In this work, we investigated the impact of the level sampling process on the ZSG capabilities of RL agents. We found adaptive sampling strategies are best understood as data regularisation techniques minimising the mutual information between the agent's internal representation and the identity of training levels. In doing so, these methods minimise an upper bound on the generalisation gap, and our experiments show that this bound acts as an effective proxy for reducing this gap in practice. This theoretical framing allowed us to understand the mechanism behind the improved generalisation achieved by value loss prioritised level sampling, which had only been justified empirically in prior work. We introduced DRED, a framework combining adaptive sampling with the generation of new levels using a learned model of the context distribution. We propose VAE-DRED, an application of DRED using a VAE to learn the context distribution. Our experiments show that VAE-DRED prevents the significant distributional shift observed in other UED methods. By jointly achieving low GenGap and ShiftGap, VAE-DRED achieves strong generalisation performance on in-distribution test levels, while also being robust to in-context edge cases.

In future work, we plan to investigate how DRED methods perform in more complex environments. Our experiments show that unsupervised environment generation can be problematic, even in gridworlds, and these issues are bound to worsen when the environment parameter space has higher complexity and dimensionality. DRED possesses the ability to leverage an existing dataset to inform its generative process, which we believe will be instrumental in scaling environment design techniques to practical applications. We are particularly interested in studying how DRED could leverage real world datasets of level parameters that have started to become available. Li et al. (2021) introduced a dataset of indoor environments geared towards robotics and embodied AI tasks, Wilson et al. (2021) published city maps for autonomous driving while Ma et al. (2023); Cao et al. (2019) published a dataset of protein docking problems. Level parameters remain costly to collect or prescribe manually, and thus these datasets remain much smaller in size than text or image datasets. In maximising the generalisation potential of a limited number of training environments, we hope DRED can reduce start-up costs associated with extending RL to new practical applications.

Impact statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

We believe parametrisable simulators are better suited to benchmark RL algorithms than procedural environments, as they provide a fine degree of control over the environment and are more consistent with a realistic application setting, as argued by Kirk et al. (2023). However, reproducibility can be challenging without access to the data generated during experiments. To assist with this, we make all of our experimental data, including model checkpoints, level datasets, logged data and the code for reproducing the figures in this paper openly available.

We open-source our code for specifying arbitrary CMDPs in Minigrid and generate their associated level sets (we describe the generation process in detail in Appendix D). We also provide a dataset of 1.5M procedurally generated minigrid base layouts to facilitate level set generation.

Acknowledgements

This work was supported by the United Kingdom Research and Innovation EPSRC Centre for Doctoral Training in Robotics and Autonomous Systems (RAS) in Edinburgh.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 29304–29320, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Agostinelli, F., McAleer, S., Shmakov, A., and Baldi, P. Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0070-z. URL https://doi.org/10.1038/s42256-019-0070-z.

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5048–5058, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html.

Azad, A. S., Gur, I., Emhoff, J., Alexis, N., Faust, A., Abbeel, P., and Stoica, I. Clutr: Curriculum learning via unsupervised task representation learning. In *International Conference on Machine Learning*, pp. 1361–1395. PMLR, 2023.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents (extended abstract). In Yang, Q. and Wooldridge, M. J. (eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 4148–4152. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/585.

Bertrán, M., Martínez, N., Phielipp, M., and Sapiro, G. Instance-based generalization in reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/82674fc29bc0d9895cee346548c2cb5c-Abstract.html.

Cao, Y., Chen, T., Wang, Z., and Shen, Y. Learning to optimize in swarms. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 15018–15028, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/ec04e8ebba7e132043e5b4832e54f070-Abstract.html.

Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on*

Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of *Proceedings of Machine Learning Research*, pp. 1282–1289. PMLR, 2019. URL http://proceedings.mlr.press/v97/cobbe19a.html.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056. PMLR, 2020. URL http://proceedings.mlr.press/v119/cobbe20a.html.

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A. M., Russell, S., Critch, A., and Levine, S. Emergent complexity and zero-shot transfer via unsupervised environment design. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

Dunion, M. and Albrecht, S. V. Multi-view disentanglement for reinforcement learning with multiple cameras. In *1st Reinforcement Learning Conference*, 2024.

Dunion, M., McInroe, T., Luck, K. S., Hanna, J., and Albrecht, S. Conditional mutual information for disentangled representations in reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Ghosh, D., Rahme, J., Kumar, A., Zhang, A., Adams, R. P., and Levine, S. Why generalization in RL is difficult: Epistemic pomdps and implicit partial observability. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 25502–25515, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/d5ff135377d39f1de7372c95c74dd962-Abstract.html.

Gumin, M. Wave function collapse algorithm. https://github.com/mxgmn/, 2016.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

Igl, M., Ciosek, K., Li, Y., Tschiatschek, S., Zhang, C., Devlin, S., and Hofmann, K. Generalization in reinforcement learning with selective noise injection and information bottleneck. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 13956–13968, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/e2ccf95a7f2e1878fcafc8376649b6e8-Abstract.html.

Jakobi, N. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, 6:325 – 368, 1997.

Jiang, M., Dennis, M., Parker-Holder, J., Foerster, J. N., Grefenstette, E., and Rocktäschel, T. Replay-guided adversarial environment design. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 1884–1897, 2021a. URL https://proceedings.neurips.cc/paper/2021/hash/0e915db6326b6fb6a3c56546980a8c93-Abstract.html.

Jiang, M., Grefenstette, E., and Rocktäschel, T. Prioritized level replay. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4940–4950. PMLR, 2021b. URL http://proceedings.mlr.press/v139/jiang21b.html.

Jiang, M., Dennis, M., Parker-Holder, J., Lupu, A., Küttler, H., Grefenstette, E., Rocktäschel, T., and Foerster, J. Grounding aleatoric uncertainty for unsupervised environment design. *Advances in Neural Information Processing Systems*, 35:32868–32881, 2022.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y. (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference*

Track Proceedings, 2014. URL http://arxiv.org/abs/1312.6114.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 1106–1114, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

Lee, J., Hwango, J., et al. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 2020.

Li, Z., Yu, T., Sang, S., Wang, S., Song, M., Liu, Y., Yeh, Y., Zhu, R., Gundavarapu, N. B., Shi, J., Bi, S., Yu, H., Xu, Z., Sunkavalli, K., Hasan, M., Ramamoorthi, R., and Chandraker, M. Openrooms: An open framework for photorealistic indoor scene datasets. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 7190–7199. Computer Vision Foundation / IEEE, 2021. doi: 10.1109/CVPR46437.2021.00711. URL https://openaccess.thecvf.com/content/CVPR2021/html/Li_OpenRooms_An_Open_Framework_for_Photorealistic_Indoor_Scene_Datasets_CVPR_2021_paper.html.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Ma, Z., Guo, H., Chen, J., Li, Z., Peng, G., Gong, Y.-J., Ma, Y., and Cao, Z. Metabox: A benchmark platform for meta-black-box optimization with reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Balcan, M. and Weinberger, K. Q. (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 1928–1937. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/mniha16.html.

Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., and Song, D. X. Assessing generalization in deep reinforcement learning. *ArXiv preprint*, abs/1810.12282, 2018. URL https://arxiv.org/abs/1810.12282.

Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J. N., Grefenstette, E., and Rocktäschel, T. Evolving curricula with regret-based environment design. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., and Sabato, S. (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17473–17498. PMLR, 2022. URL https://proceedings.mlr.press/v162/parker-holder22a.html.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2817–2826. PMLR, 2017. URL http://proceedings.mlr.press/v70/pinto17a.html.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. Automatic data augmentation for generalization in reinforcement learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 5402–5415, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/2b38c2df6a49b97f706ec9148ce48d86-Abstract.html.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pp. 1278–1286. JMLR.org, 2014. URL http://proceedings.mlr.press/v32/rezende14.html.

Rudin, N., Hoeller, D., Reist, P., and Hutter, M. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pp. 91–100. PMLR, 2022.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05952.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv*, 2017.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

White, T. Sampling generative networks. *ArXiv preprint*, abs/1609.04468, 2016. URL https://arxiv.org/abs/1609.04468.

Wilson, B., Qi, W., Agarwal, T., Lambert, J., Singh, J., Khandelwal, S., Pan, B., Kumar, R., Hartnett, A., Kaesemodel Pontes, J., Ramanan, D., Carr, P., and Hays, J. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In Vanschoren, J. and Yeung, S. (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/4734ba6f3de83d861c3176a6273cac6d-Paper-round2.pdf.

Xu, A. and Raginsky, M. Information-theoretic analysis of generalization capability of learning algorithms. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 2524–2533, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/ad71c82b22f4f65b9398f76d8be4c615-Abstract.html.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yarats, D., Kostrikov, I., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=GY6-6sTvGaf.

Zhang, C., Vinyals, O., Munos, R., and Bengio, S. A study on overfitting in deep reinforcement learning. *ArXiv preprint*, abs/1804.06893, 2018. URL https://arxiv.org/abs/1804.06893.

Zhang, Y., Abbeel, P., and Pinto, L. Automatic curriculum learning through value disagreement. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/566f0ea4f6c2e947f36795c8f58ba901-Abstract.html.

A. Theoretical results

Lemma A.1. *Given a set of training levels L and an agent model $\pi = f \circ b$, where $b(H_t^o) = h_t \in \mathbb{B}$ is an intermediate representation function and $f : \mathbb{B} \to \Delta^{\mathbb{A}}$ maps to the agent's action distribution, we have*

$$I(L;\pi) \leq \mathbf{H}(\mathrm{i}) + \sum_{i \in L} \int_{\mathbb{B}} p(h,i) \log p(i|h) dh, \tag{11}$$

where the right-hand side is equivalent to $I(L;b)$.

proof:

Since the information chain of our model follows $H_t^o \to b \to f$, we have $I(L; f \circ b) \leq I(L;b)$, from the data processing inequality. $I(L;b)$ can then be manipulated as follows

$$I(L;b) = \sum_{i \in L} \int_{\mathbb{B}} p(h,i) \log \frac{p(h,i)}{p(h)p(i)} dh \tag{12}$$

$$= -\sum_{i \in L} \int_{\mathbb{B}} p(h,i) \log p(i) dh + \sum_{i \in L} \int_{\mathbb{B}} p(h,i) \log p(i|h) dh \tag{13}$$

$$= \mathbf{H}(\mathrm{i}) + \sum_{i \in L} \int_{\mathbb{B}} p(h,i) \log p(i|h) dh. \tag{14}$$

B. Procgen additional experimental results

B.1. Measuring the relationship between $\mathrm{I}(L;b)$ and the GenGap across sampling methods and procgen games

Table 1: $\mathrm{I}(L;b)$ measured under different adaptive sampling strategies. We report aggregated $\mathrm{I}(L;b)$ across 5 training runs, each initialised with a different seed. To compute $\mathrm{I}(L;b)$ for each run and environment, we first fit a linear classifier $p_{\boldsymbol{\theta}}$ to predict level identities from the agent's penultimate layer outputs, using rollouts from levels sampled uniformly from L. We then estimate $\mathrm{I}(L;b)$ using Equation (6), using a different set of rollouts also sampled uniformly from L. In the last row we compute the mean $\mathrm{I}(L;b)$ across environments for each run, and we report the mean and standard deviation of that quantity across all runs. Bolded methods are not significantly different from the method with lowest mean ($p < 0.05$), unless all are, in which case none are bolded.

Environment	$S = S^V$	$P_S = \mathcal{U}$	$S = S^{\mathrm{MI}}$	$S = S^V, P_{S'} = \mathcal{U}$	$S = S^V, S' = S^{\mathrm{MI}}$
Bigfish	**1.74 ± 0.36**	4.33 ± 0.48	**1.45 ± 0.36**	2.43 ± 0.53	**1.77 ± 0.70**
Heist	4.67 ± 0.31	**3.68 ± 0.36**	**4.08 ± 0.50**	4.54 ± 0.37	4.76 ± 0.19
Climber	4.03 ± 0.23	4.40 ± 0.18	**2.44 ± 0.44**	4.48 ± 0.15	4.09 ± 0.17
Caveflyer	3.01 ± 0.17	3.94 ± 0.21	**1.72 ± 0.20**	3.38 ± 0.14	3.00 ± 0.18
Jumper	4.49 ± 0.13	**3.87 ± 0.40**	**3.38 ± 0.38**	4.37 ± 0.19	4.39 ± 0.07
Fruitbot	**0.15 ± 0.09**	2.76 ± 0.11	0.40 ± 0.17	0.24 ± 0.09	**0.09 ± 0.07**
Plunder	1.46 ± 0.22	2.95 ± 0.74	**1.09 ± 0.20**	1.80 ± 0.41	1.76 ± 0.22
Coinrun	**1.38 ± 0.07**	2.29 ± 0.19	**1.30 ± 0.14**	1.59 ± 0.05	**1.36 ± 0.15**
Ninja	2.36 ± 0.30	2.62 ± 0.24	**1.42 ± 0.44**	3.00 ± 0.17	2.45 ± 0.19
Leaper	1.72 ± 0.08	1.06 ± 0.08	**0.79 ± 0.13**	1.79 ± 0.22	1.65 ± 0.18
Maze	4.76 ± 0.09	**4.27 ± 0.22**	4.79 ± 0.15	4.73 ± 0.09	4.72 ± 0.04
Miner	**4.36 ± 0.13**	4.81 ± 0.02	**4.42 ± 0.27**	**4.53 ± 0.22**	**4.29 ± 0.17**
Dodgeball	3.88 ± 0.37	2.51 ± 0.32	**0.89 ± 0.34**	4.05 ± 0.19	3.54 ± 0.41
Starpilot	**1.10 ± 0.06**	2.07 ± 0.14	1.38 ± 0.11	1.44 ± 0.13	**1.20 ± 0.08**
Chaser	**1.37 ± 0.12**	3.18 ± 0.26	**1.36 ± 0.14**	1.98 ± 0.34	**1.43 ± 0.25**
Bossfight	**1.33 ± 0.12**	4.19 ± 0.35	**1.17 ± 0.12**	**1.15 ± 0.17**	**1.16 ± 0.33**
Average Mutual Information	2.61 ± 0.05	3.31 ± 0.03	**2.00 ± 0.07**	2.84 ± 0.08	2.60 ± 0.02

To better understand the interaction between the mutual information, the value loss and the generalisation gap, we plot our estimate for $\mathrm{I}(L;b)$ at the end of training against the GenGap and the ℓ_1-value loss for all methods tested and across Procgen games in Figure 7. We find a positive correlation (correlation coefficient $\rho = 0.60$) and rank correlation (Kendall

Table 2: Level classifier accuracies measured under different adaptive sampling strategies. We report aggregated accuracies across 5 training runs, each initialised with a different seed. To compute this quantity for each run and environment, we first fit a linear classifier $p_{\boldsymbol{\theta}}$ to predict level identities from the agent's penultimate layer outputs, using rollouts from levels sampled uniformly from L. We then measure the classifier accuracy, using a different set of rollouts also sampled uniformly from L. In the last row we compute the mean accuracy across environments for each run, and we report the mean and standard deviation of that quantity across all runs. Bolded methods are not significantly different from the method with lowest mean ($p < 0.05$), unless all are, in which case none are bolded.

ENVIRONMENT	$S = S^V$	$P_S = \mathcal{U}$	$S = S^{\text{MI}}$	$S = S^V, P_{S'} = \mathcal{U}$	$S = S^V, S' = S^{\text{MI}}$
BIGFISH	**17.2 ± 10.1**	66.5 ± 5.5	**12.8 ± 4.5**	**24.7 ± 10.3**	**14.3 ± 9.9**
HEIST	83.1 ± 9.5	77.8 ± 3.8	**38.1 ± 20.7**	77.5 ± 5.6	72.5 ± 9.1
CLIMBER	53.4 ± 15.0	82.9 ± 3.3	**17.4 ± 10.5**	73.2 ± 3.6	64.0 ± 5.6
CAVEFLYER	34.9 ± 5.4	62.5 ± 3.9	**21.4 ± 1.5**	48.5 ± 6.0	**33.1 ± 11.5**
JUMPER	62.0 ± 10.4	64.4 ± 6.3	**28.0 ± 17.8**	64.1 ± 5.6	59.4 ± 10.0
FRUITBOT	**2.7 ± 4.1**	18.0 ± 1.3	**5.3 ± 6.9**	**2.6 ± 2.0**	**2.3 ± 2.8**
PLUNDER	**13.5 ± 9.8**	25.3 ± 8.8	**2.6 ± 3.4**	16.1 ± 6.8	**15.5 ± 9.9**
COINRUN	**14.1 ± 0.8**	22.3 ± 1.9	**10.5 ± 4.7**	**11.6 ± 4.0**	**13.4 ± 4.2**
NINJA	22.8 ± 7.5	33.9 ± 2.4	**5.6 ± 7.2**	27.8 ± 2.9	18.4 ± 4.9
LEAPER	10.2 ± 5.6	11.6 ± 2.1	8.5 ± 4.0	13.0 ± 1.0	12.9 ± 3.2
MAZE	69.7 ± 2.6	**65.5 ± 1.3**	**69.2 ± 10.2**	**63.9 ± 3.8**	**67.7 ± 7.8**
MINER	85.5 ± 1.1	92.0 ± 0.7	**67.7 ± 8.2**	80.4 ± 4.3	**78.1 ± 4.5**
DODGEBALL	63.1 ± 0.8	45.4 ± 7.3	**5.7 ± 5.8**	68.5 ± 7.5	53.5 ± 7.4
STARPILOT	8.1 ± 3.1	14.7 ± 3.3	**3.2 ± 2.4**	9.0 ± 2.7	6.8 ± 1.1
CHASER	**13.0 ± 3.3**	40.4 ± 4.7	**7.8 ± 4.0**	27.7 ± 12.3	**16.2 ± 6.1**
BOSSFIGHT	**6.9 ± 6.1**	60.2 ± 14.3	**3.1 ± 4.1**	**10.4 ± 6.2**	**10.5 ± 8.5**
AVERAGE CLASSIFIER ACCURACY	35.0 ± 1.7	49.0 ± 0.9	**19.2 ± 2.2**	38.7 ± 1.3	33.7 ± 1.6

rank correlation coefficient $\xi = 0.50, p < 1e-50$) between $\text{I}(L; b)$ and the GenGap. We find similar correlation ($\rho = 0.50$) and rank correlation ($\xi = 0.49, p < 1e-47$) between $\text{I}(L; b)$ and the normalised GenGap. We also observe a weaker but statistically significant negative correlation ($\rho = -0.18$) and negative rank correlation ($\xi = -0.11, p < 0.001$) between $\text{I}(L; b)$ and the ℓ_1-value loss.

We report the $\text{I}(L; b)$ averaged across seeds for all games and methods tested in Table 1. In order to obtain a more intuitive quantification of $\text{I}(L; b)$, we also report the classification accuracy of the linear classifier $p_{\boldsymbol{\theta}}$ in Table 2, as these two quantities are proportional to one-another. Out of 200 training levels, the classifier correctly predicts the current level 49% of the times under uniform sampling, 35% under $(S = S^V)$ and 19% under S^{MI}. While adaptive sampling strategies are able to significantly reduce $\text{I}(L; b)$, the mean classifier accuracy is still 70 times random guessing for $(S = S^V)$ and 38 times random guessing for $(S = S^{\text{MI}})$.

B.2. A qualitative analysis of when adaptive sampling strategies may or may not be effective in reducing the GenGap

We report in Figure 8 that, when compared to uniform sampling across procgen games, adaptive sampling strategies are significantly more likely to reduce the GenGap. Strategies employing $(S = S^V)$ as their primary scoring function are also more likely to improve their test set scores. However, we measure significant variability across procgen games for $\text{I}(L; b)$ in Table 1 (and by extension for the GenGap) for the different strategies tested. To better understand why, we compare the measured accuracy with a qualitative analysis of the observations and levels encountered in the "Maze" and "Bigfish" games (see Figure 9 for renderings of different levels from each game). In Maze, the classifier accuracy remains over 60% ($120\times$ random) for all methods tested and the reduction in GenGap is insignificant. On the other hand, in Bigfish all adaptive sampling strategies tested lead to a significant reduction in classifier accuracy when compared to uniform sampling, dropping from 65% to between 12% and 25% (depending on the strategy), and correspond to a significant drop in the GenGap and improvement in test scores.

In Maze, the observation space lets the agent observe the full layout at every timestep. The maze layout is unique to each level and should be easily identifiable by the agent's ResNet architecture. Intuitively, we can hypothesise that adaptive sampling strategies will not be effective if all the levels are easily identifiable by the model, which appears to be the case in Maze. In these cases, other data regularisation techniques, such as augmenting the observations, could be more effective,

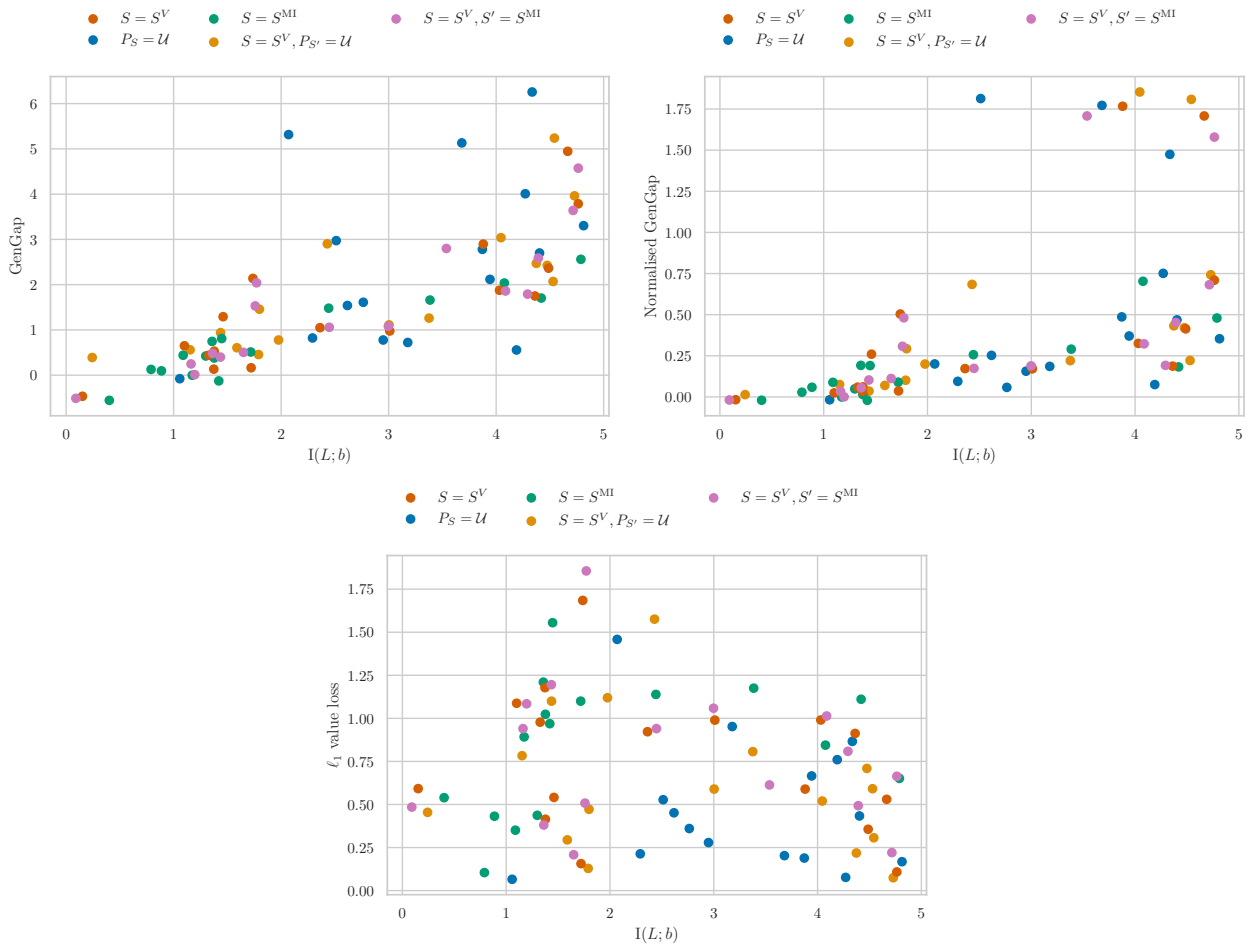


Figure 7: Scatter plot displaying how I$(L; b)$ compares to the GenGap (top left), to the normalised GenGap (top right), and to the ℓ_1 average value loss (bottom) across all methods and Procgen games, at the end of training. Each plotted point represents the average of 5 seeds in a particular game.

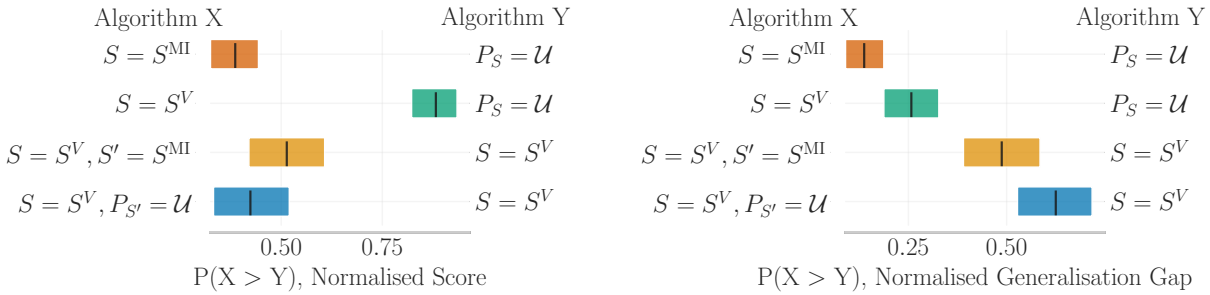


Figure 8: Probability of algorithm X incurring a higher normalised test score (left) and GenGap (right) than algorithm Y. Colored boxes indicate the 95% confidence interval.

and in fact (Jiang et al., 2021b) report that Maze is one of the games where combining PLR with UCB-DrAC (Raileanu et al., 2021), a data augmentation method, leads to a significant improvement in test scores.

On the other hand, we observe that many of the Bigfish levels yield similar observations. Indeed, both the features relevant to the task (the fish) and irrelevant (the background) are similar in many of the training levels. Furthermore, there's significant

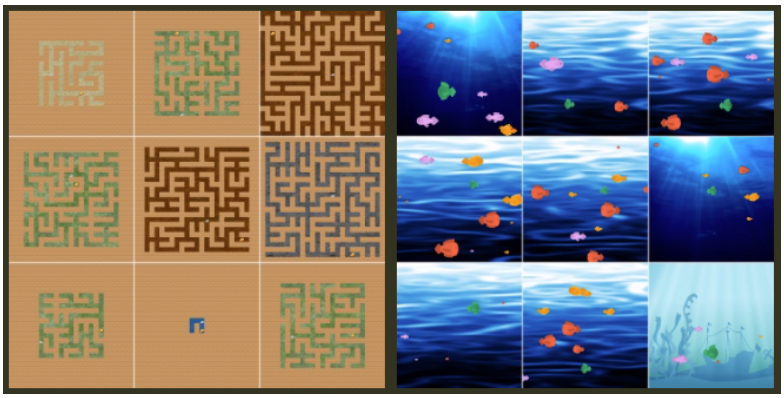


Figure 9: Agent observations sampled from 9 levels from the Maze (left) and Bigfish (right) games of the Procgen benchmark.

variation in the observations encountered during an episode, as fish constantly appear and leave the screen. Yet, some levels (top left, middle and bottom right) are easily identifiable thanks to their background, and we can expect them to be more prone to overfitting. By using the background to identify the current level the agent is able to infer when and where fish will appear on screen. This is an effective strategy to solve the training level in question and to accurately predict its value function, however applying this strategy to unseen levels at test time will fail, as the background and the fish locations are not correlated across levels. Adaptive sampling strategies minimising $\mathrm{I}(L; b)$ de-prioritise problematic levels when the agent's representation starts overfitting, essentially performing data regularisation via a form of rejection sampling.

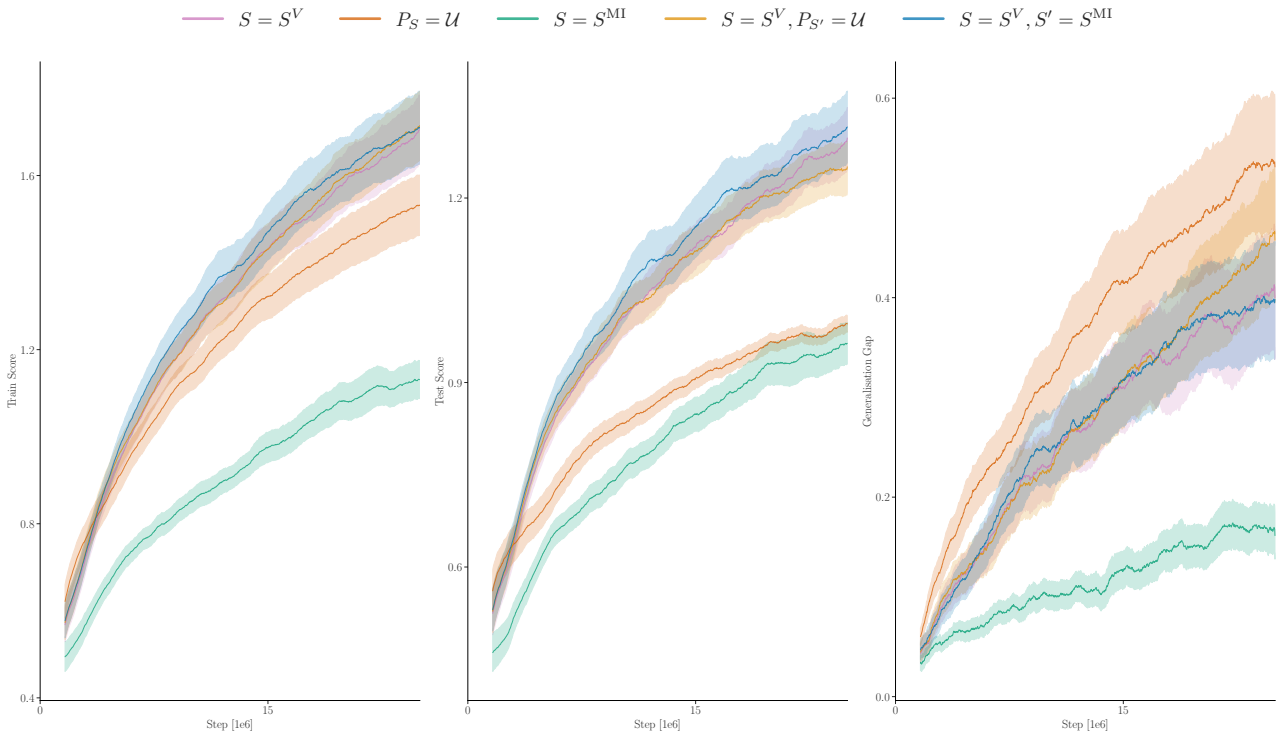
B.3. Procgen extended results

Table 3: Test scores of a PPO agent trained under different adaptive sampling strategies. We report aggregated scores across 5 training runs, each initialised with a different seed. For each run the test score is obtained by evaluating the final policy's average score on 1000 episodes, each episode sampling a different level not in the train set. Following (Raileanu et al., 2021), normalised test scores per run are computed by dividing its test score per run for each environment by the corresponding average test score of the uniform-sampling strategy over all runs. In the last row we compute the mean normalised score across environments for each run, and we report the mean and standard deviation of that quantity across all runs. Bolded methods are not significantly different from the method with highest mean ($p < 0.05$), unless all are, in which case none are bolded.

ENVIRONMENT	$S = S^V$	$P_S = \mathcal{U}$	$S = S^{\mathrm{MI}}$	$S = S^V, P_{S'} = \mathcal{U}$	$S = S^V, S' = S^{\mathrm{MI}}$
BIGFISH	**11.5 ± 2.1**	4.2 ± 1.2	7.3 ± 0.6	**10.2 ± 2.0**	**12.4 ± 3.4**
HEIST	3.2 ± 0.7	2.9 ± 0.2	2.7 ± 0.5	2.9 ± 0.8	3.0 ± 0.4
CLIMBER	**6.9 ± 0.3**	5.8 ± 0.3	4.9 ± 0.6	**7.0 ± 0.4**	**7.0 ± 0.2**
CAVEFLYER	**6.3 ± 0.1**	5.7 ± 0.1	5.8 ± 0.4	**6.5 ± 0.3**	**6.2 ± 0.2**
JUMPER	**6.0 ± 0.1**	5.7 ± 0.2	5.8 ± 0.1	**6.0 ± 0.0**	**5.9 ± 0.1**
FRUITBOT	**28.4 ± 0.4**	**27.7 ± 0.8**	23.6 ± 1.4	**28.2 ± 0.5**	**28.6 ± 0.8**
PLUNDER	**7.3 ± 1.0**	5.0 ± 0.3	5.6 ± 1.5	**7.7 ± 1.4**	**8.5 ± 0.8**
COINRUN	**8.9 ± 0.2**	8.7 ± 0.2	**9.1 ± 0.1**	**9.0 ± 0.1**	**9.1 ± 0.1**
NINJA	**7.1 ± 0.4**	6.1 ± 0.2	5.0 ± 0.6	**7.1 ± 0.2**	**7.2 ± 0.3**
LEAPER	**7.1 ± 0.4**	4.5 ± 0.1	2.7 ± 0.1	**6.8 ± 1.5**	**7.2 ± 1.8**
MAZE	**5.6 ± 0.4**	**5.3 ± 0.3**	5.0 ± 0.1	**5.4 ± 0.4**	**5.6 ± 0.5**
MINER	**9.6 ± 0.3**	**9.3 ± 0.3**	8.6 ± 0.5	**9.6 ± 0.3**	**9.7 ± 0.3**
DODGEBALL	**2.3 ± 0.3**	1.6 ± 0.2	0.9 ± 0.1	**2.1 ± 0.3**	**2.2 ± 0.4**
STARPILOT	**26.9 ± 1.3**	**26.5 ± 2.0**	18.9 ± 0.9	**25.2 ± 2.3**	**26.7 ± 2.4**
CHASER	**6.6 ± 0.9**	3.9 ± 1.1	**5.2 ± 0.8**	**6.0 ± 1.1**	**5.9 ± 1.5**
BOSSFIGHT	**9.0 ± 0.3**	7.4 ± 0.3	7.8 ± 0.4	8.4 ± 0.3	**8.6 ± 0.5**
NORMALISED TEST SCORES (%)	**130.3 ± 5.3**	100.0 ± 1.4	97.0 ± 4.0	**125.4 ± 5.3**	**131.3 ± 8.4**

Table 4: Train scores of a PPO agent trained under different adaptive sampling strategies. We report aggregated scores across 5 training runs, each initialised with a different seed. For each run the train score is obtained by evaluating the final policy's average score on 1000 episodes, each episode sampling a different level from the train set. Following (Raileanu et al., 2021), normalised train scores per run are computed by dividing its train score per run for each environment by the corresponding average test score of the uniform-sampling strategy over all runs. In the last row we compute the mean normalised score across environments for each run, and we report the mean and standard deviation of that quantity across all runs. Bolded methods are not significantly different from the method with highest mean ($p < 0.05$), unless all are, in which case none are bolded.

ENVIRONMENT	$S = S^V$	$P_S = \mathcal{U}$	$S = S^{\text{MI}}$	$S = S^V, P_{S'} = \mathcal{U}$	$S = S^V, S' = S^{\text{MI}}$
BIGFISH	**13.6 ± 0.9**	**10.5 ± 1.8**	8.2 ± 1.3	**13.1 ± 1.4**	**14.4 ± 3.0**
HEIST	**8.2 ± 0.5**	**8.0 ± 0.6**	4.7 ± 0.7	**8.2 ± 0.7**	**7.6 ± 0.5**
CLIMBER	8.8 ± 0.2	8.5 ± 0.3	6.4 ± 0.8	**9.4 ± 0.3**	8.9 ± 0.2
CAVEFLYER	7.3 ± 0.2	**7.8 ± 0.3**	6.3 ± 0.1	**7.7 ± 0.2**	7.3 ± 0.1
JUMPER	**8.4 ± 0.2**	**8.5 ± 0.1**	7.4 ± 0.3	**8.5 ± 0.1**	**8.5 ± 0.2**
FRUITBOT	27.9 ± 0.4	**29.3 ± 0.3**	23.1 ± 0.7	28.6 ± 0.3	28.1 ± 0.3
PLUNDER	**8.6 ± 1.0**	5.7 ± 0.4	6.1 ± 1.3	**9.2 ± 1.7**	**10.1 ± 0.9**
COINRUN	9.5 ± 0.1	9.5 ± 0.1	9.5 ± 0.1	9.6 ± 0.0	9.6 ± 0.1
NINJA	**8.1 ± 0.2**	7.6 ± 0.2	4.9 ± 0.3	**8.2 ± 0.1**	**8.3 ± 0.2**
LEAPER	**7.2 ± 0.1**	4.4 ± 0.1	2.8 ± 0.1	**7.3 ± 1.6**	**7.7 ± 1.9**
MAZE	**9.4 ± 0.1**	**9.3 ± 0.1**	7.5 ± 0.3	**9.4 ± 0.1**	**9.3 ± 0.1**
MINER	11.3 ± 0.3	**12.6 ± 0.1**	10.3 ± 0.4	11.6 ± 0.2	11.5 ± 0.2
DODGEBALL	**5.2 ± 0.4**	**4.6 ± 0.6**	1.0 ± 0.3	**5.2 ± 0.6**	**5.0 ± 0.6**
STARPILOT	27.6 ± 1.8	**31.8 ± 1.5**	19.3 ± 0.9	26.2 ± 1.8	26.7 ± 1.9
CHASER	**6.8 ± 0.9**	4.6 ± 1.3	**6.0 ± 1.2**	**6.8 ± 1.0**	**6.3 ± 1.1**
BOSSFIGHT	**9.4 ± 0.2**	8.0 ± 0.4	7.8 ± 0.1	8.9 ± 0.3	8.9 ± 0.3
NORMALISED TRAIN SCORES (%)	**170.4 ± 2.0**	153.1 ± 2.1	113.1 ± 5.0	**171.4 ± 5.9**	**171.1 ± 9.4**



(a) Train Score (b) Test Score (c) Generalisation Gap

Figure 10: Procgen normalised scores across environments and generalisation gap over the course of training.

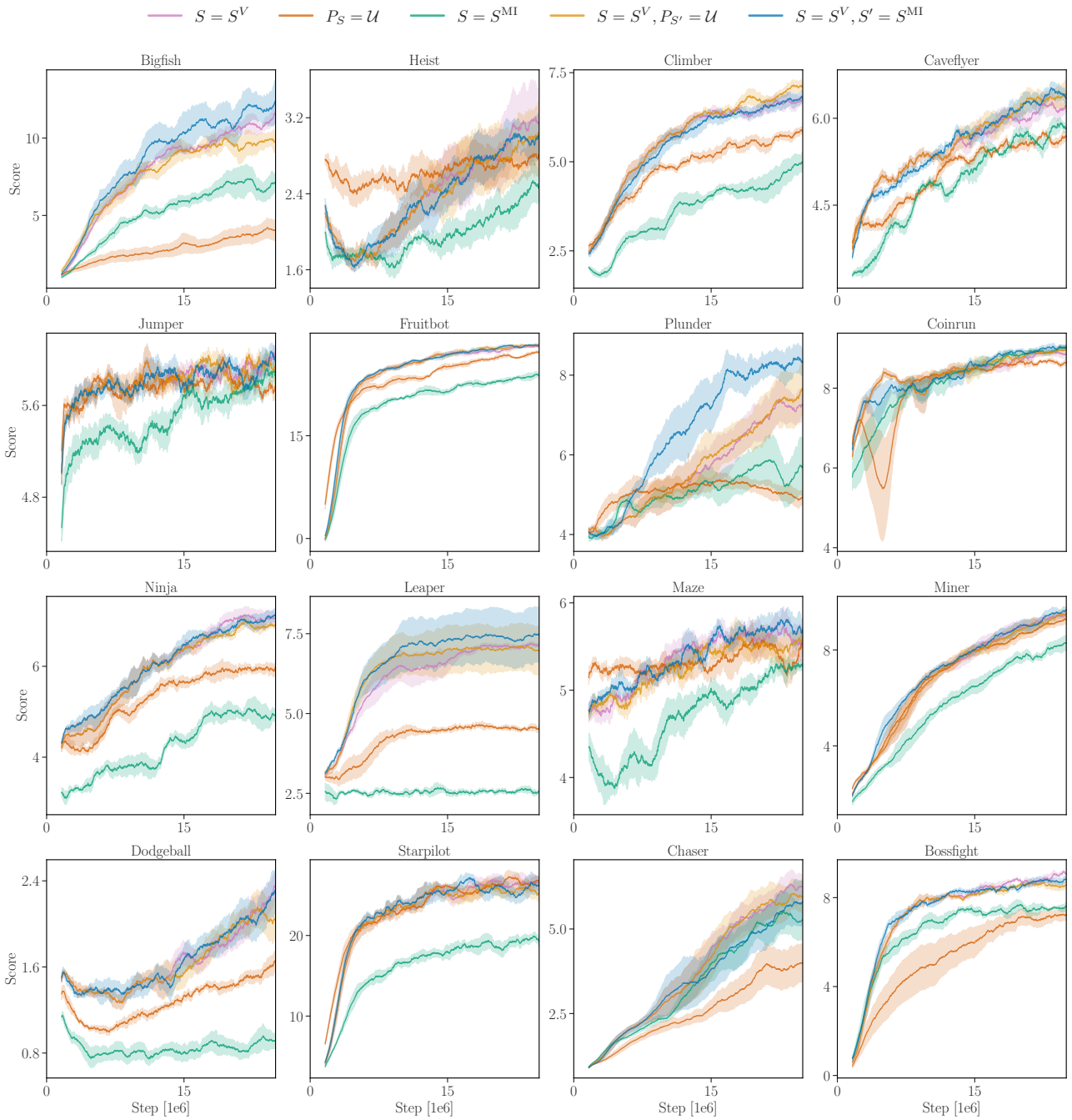


Figure 11: Procgen per game test set scores over the course of training.

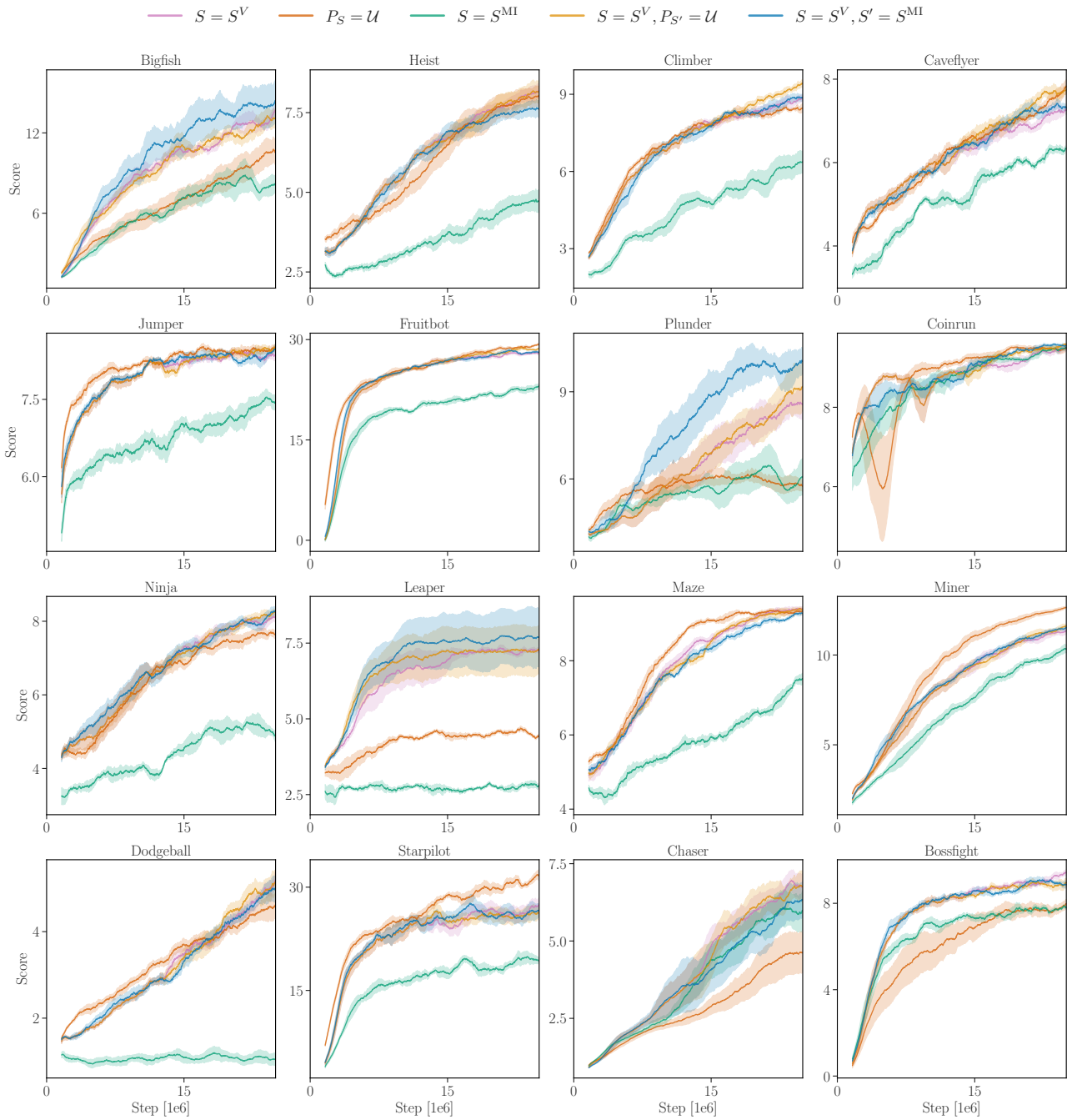


Figure 12: Procgen per game train set scores over the course of training.

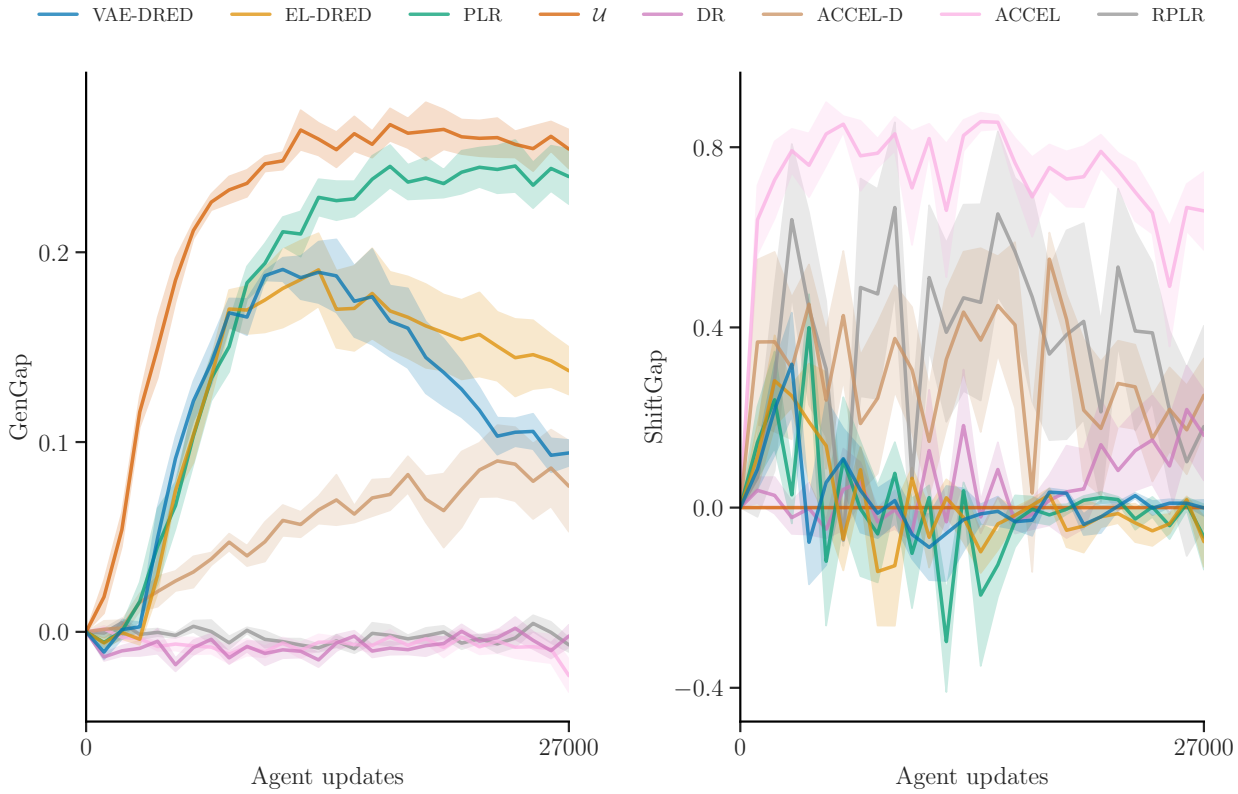
C. Minigrid additional experimental results

C.1. Quantifying the distributional shift in minigrid

In Figure 13, we report the GenGap and the ShiftGap in the Minigrid experiments. The GenGap is not helpful in interpreting the poor test set performance of UED methods. In fact, we report in Figure 15 that UED methods tend to perform well when evaluated on levels sampled from P_Λ, while performing poorly on the train and test sets. This results in the GenGap remaining close to zero throughout training.

We would also expect the GenGap and the train and test scores to be near-zero for a random policy, that is if the agent was not learning anything at all. However ACCEL and RPLR experience a different failure mode: the UED-trained agents are learning, as demonstrated by their scores on their respective training distributions, but they are learning to solve a different CMDP. In contrast, the ShiftGap is effective at distinguishing between the agent not learning at all and it learning an out-of-context policy.

While the ShiftGap allows us to quantify how the distributional shift impacts the agent performance, we desire to quantify the distributional shift directly, and within a feature space consistent with the CMDP semantics. We first compute the distribution $c(t, d|i_{\boldsymbol{x}})$, the probability of tile type t occurring at shortest path d from the goal location, for each level $i_{\boldsymbol{x}} \in \Lambda$. c_p is the marginal $c_p = \mathbb{E}_{i_{\boldsymbol{x}} \sim p(i_{\boldsymbol{x}})}[c(t, d|i_{\boldsymbol{x}})]$ and we measure the Jensen-Shannon Divergence $\text{JSD}(c_p||c_q)$, with $p = P_\Lambda$ and $q = \mathcal{U}(X_{\text{train}})$.



(a) (b)

Figure 13: Evolution of (a) the GenGap (Equation (2)) and (b) the ShiftGap (Equation (9)) over the course of training. Fixed set sampling strategies experience higher GenGap and low ShiftGap, while UED methods follow the opposite trend. Both DRED methods maintain near-zero ShiftGap while achieving a significantly smaller GenGap than fixed set strategies. DR spends the majority of training with a low GenGap and ShiftGap, but a sharp increase towards the end of training brings its ShiftGap to similar levels as RPLR and ACCEL-D.

We report how the JSD evolves over the course of training for different methods in Figure 14a. We observe that distributional shift occurs early on during training and remains relatively stable afterwards in all methods. Surprisingly, both DRED methods demonstrate a smaller JSD than PLR, even though PLR only samples X_{train} levels. JSD and ShiftGap are positively correlated for all methods except DR and PLR, which both present high JSD but low ShiftGap. VAE-DRED is the only generative method to maintain a low JSD and ShiftGap throughout training. Interestingly, and with the exception of $\mathcal{U}$, methods rank nearly identically to their test set performance ranking when sorted according to their JSD (lowest first). This relationship appears to hold throughout training, as can be observed by comparing the JSD with X_{test} scores in Figure 15.

In Figure 14, we report additional metrics on the levels sampled by each method. Over the course of training, only

VAE-DRED and PLR stay relatively consistent with X_{train} across the three metrics considered.

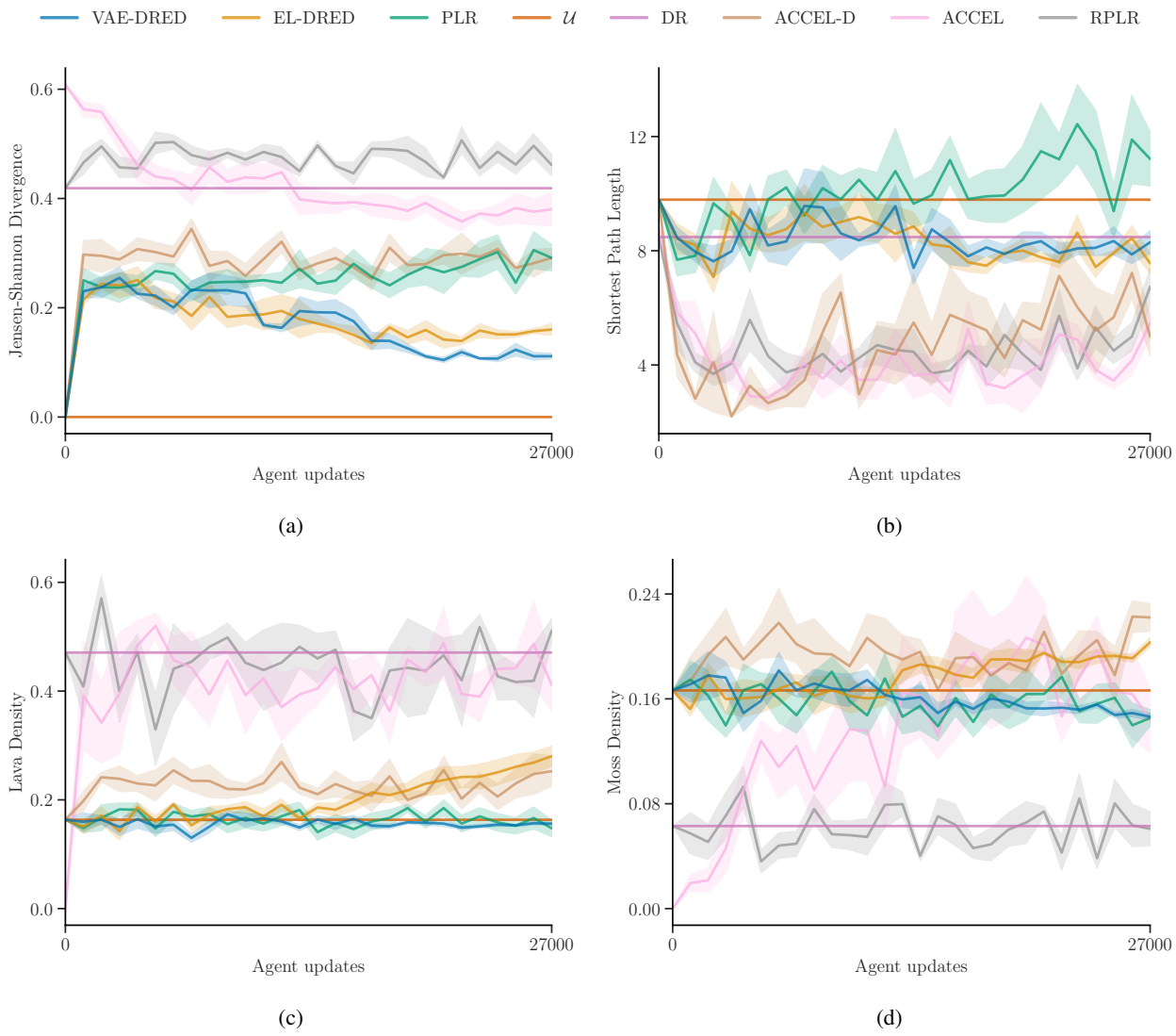


Figure 14: In sampled levels over the course of training, evolution of (a) the Jensen-Shannon divergence, (b) the shortest path length between the start and goal location, (c) the lava tile density (over non-navigable tiles) and (d) the moss tile density (over navigable tiles).

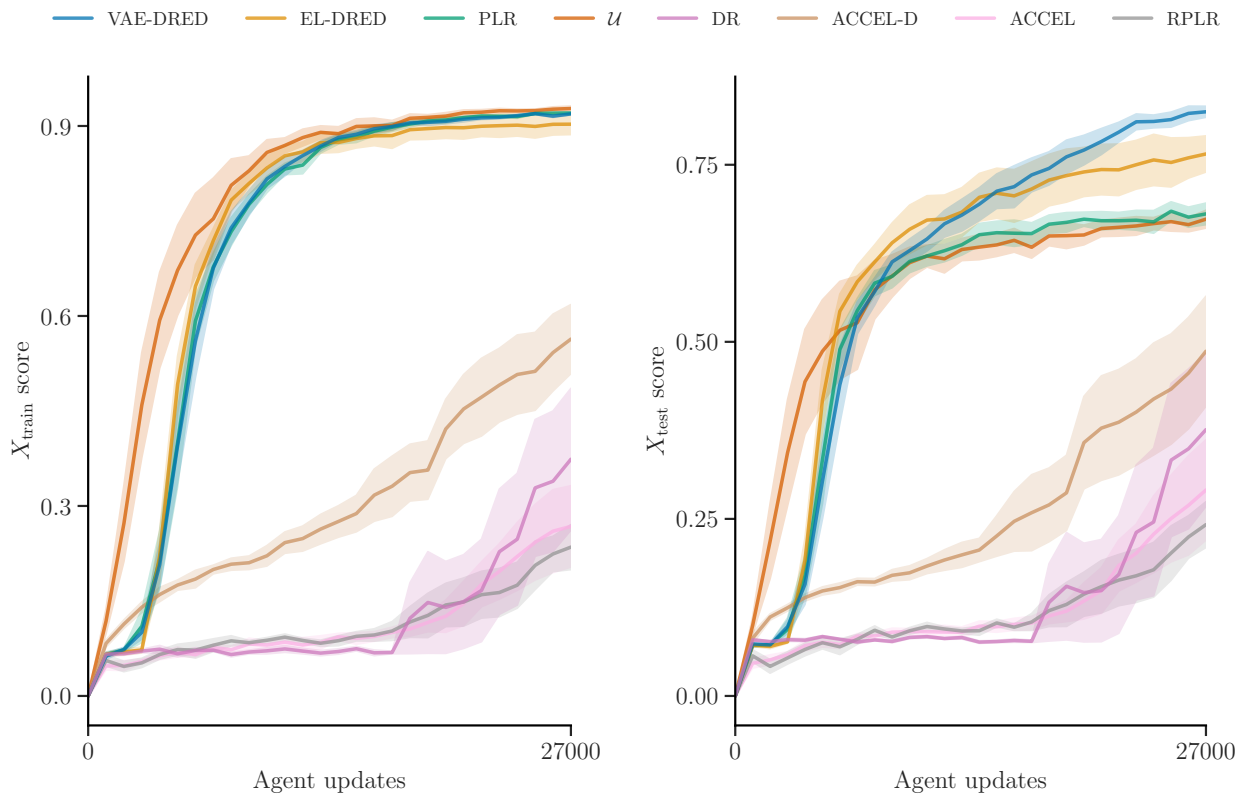


Figure 15: Minigrid scores for X_{train} and X_{test} over the course of training. Shaded area represents the standard error across 5 seeds.

C.2. Extended Minigrid results

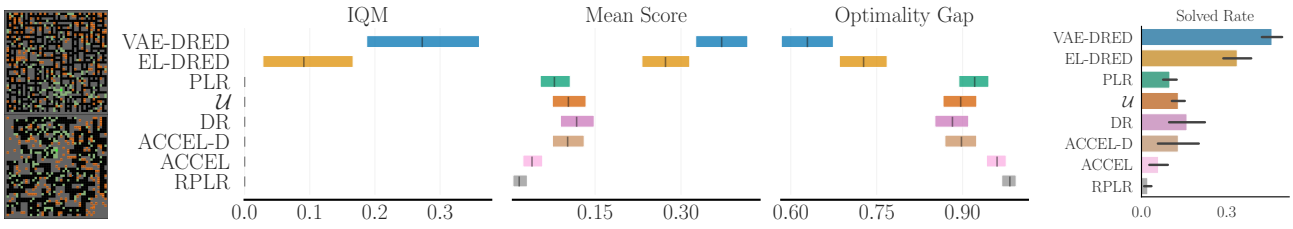


Figure 16: Aggregate final performance and mean solved rate on 216 "Hardcore" levels, with a 45x45 layout size, making Hardcore layouts 9 times larger than X_{train} layouts. Example layouts from each evaluation set are plotted on the left. The coloured boxes indicate a 99% confidence interval and the black horizontal bars indicate standard error across 5 training seeds. We refer the reader to Appendix D for additional details on our evaluation sets.

Table 5: Mean scores achieved in the Minigrid setting for each level evaluation dataset. We report the mean and standard deviation across 5 training runs, each corresponding to a different random seed. For each run the mean score is obtained by evaluating the final policy's once in each level of the corresponding level dataset, and averaging. Bolded methods are not significantly different from the method with highest mean ($p < 0.05$), unless all are, in which case none are bolded.

LEVEL DATASETS	VAE-DRED	EL-DRED	PLR	$\mathcal{U}$	DR	ACCEL-D	ACCEL	RPLR
X_{train}	**0.92 ± 0.00**	**0.90 ± 0.03**	**0.92 ± 0.00**	**0.93 ± 0.01**	0.37 ± 0.22	0.56 ± 0.11	0.27 ± 0.13	0.24 ± 0.07
X_{test}	**0.82 ± 0.02**	**0.77 ± 0.05**	0.68 ± 0.03	0.67 ± 0.03	0.38 ± 0.22	0.49 ± 0.16	0.29 ± 0.14	0.24 ± 0.06
EDGE C	**0.74 ± 0.05**	**0.67 ± 0.11**	0.35 ± 0.11	0.32 ± 0.04	0.37 ± 0.21	0.34 ± 0.17	0.25 ± 0.14	0.17 ± 0.05
HARDCORE	**0.37 ± 0.06**	**0.27 ± 0.07**	0.08 ± 0.04	0.10 ± 0.04	0.12 ± 0.09	0.10 ± 0.11	0.04 ± 0.04	0.02 ± 0.02

Table 6: Solved rates achieved in the Minigrid setting for each level evaluation dataset. We report the mean and standard deviation across 5 training runs, each corresponding to a different random seed. For each run the solved rate is obtained by counting how many levels were solved (i.e. the agent reached the goal) after evaluating the final policy's once in each level of the corresponding level dataset. Bolded methods are not significantly different from the method with highest mean ($p < 0.05$), unless all are, in which case none are bolded.

LEVEL DATASETS	VAE-DRED	EL-DRED	PLR	$\mathcal{U}$	DR	ACCEL-D	ACCEL	RPLR
X_{train}	**1.00 ± 0.00**	**0.98 ± 0.03**	**1.00 ± 0.00**	**0.99 ± 0.00**	0.48 ± 0.27	0.66 ± 0.15	0.34 ± 0.18	0.28 ± 0.09
X_{test}	**0.95 ± 0.01**	**0.89 ± 0.06**	0.79 ± 0.03	0.77 ± 0.02	0.49 ± 0.26	0.57 ± 0.20	0.36 ± 0.19	0.29 ± 0.08
EDGE C	**0.87 ± 0.06**	**0.80 ± 0.14**	0.45 ± 0.12	0.40 ± 0.05	0.46 ± 0.25	0.41 ± 0.22	0.31 ± 0.19	0.19 ± 0.06
HARDCORE	**0.46 ± 0.07**	**0.34 ± 0.10**	0.10 ± 0.05	0.13 ± 0.05	0.16 ± 0.13	0.13 ± 0.14	0.06 ± 0.07	0.02 ± 0.03

Table 7: GenGap and ShiftGap at the end of training for each method tested. We report their mean and standard deviation across 5 training runs, each corresponding to a different random seed. Bolded methods are not significantly different from the method with lowest mean ($p < 0.05$), unless all are, in which case none are bolded.

METRIC	VAE-DRED	EL-DRED	PLR	$\mathcal{U}$	DR	ACCEL-D	ACCEL	RPLR
GENGAP	0.09 ± 0.01	0.14 ± 0.03	0.24 ± 0.03	0.25 ± 0.02	**-0.00 ± 0.01**	0.08 ± 0.05	**-0.02 ± 0.02**	**-0.01 ± 0.01**
SHIFTGAP	**0.00 ± 0.04**	**-0.08 ± 0.11**	**-0.07 ± 0.14**	N/A (0)	**0.16 ± 0.20**	0.25 ± 0.16	0.66 ± 0.17	**0.18 ± 0.45**
JSD	**0.11 ± 0.01**	0.16 ± 0.02	0.29 ± 0.05	N/A (0)	0.42 ± 0.00	0.29 ± 0.03	0.38 ± 0.06	0.46 ± 0.04

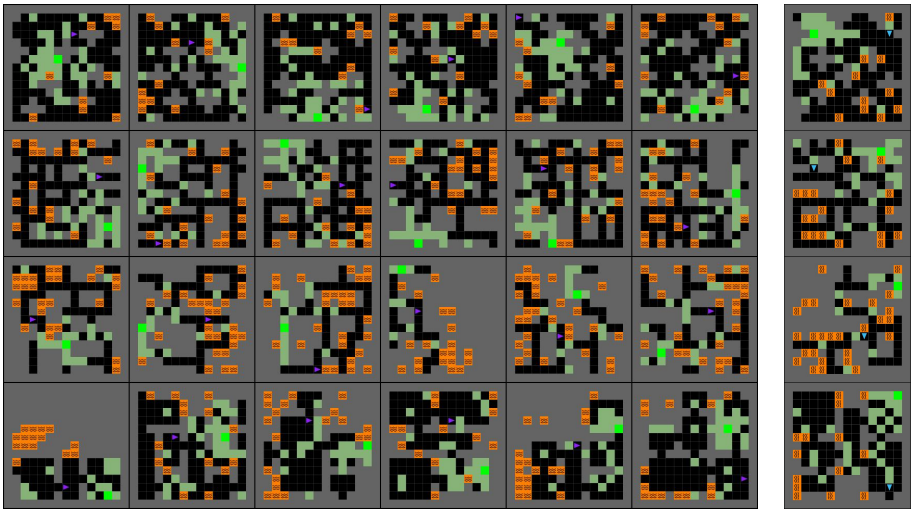
D. CMDP specification and generation

In Minigrid (Chevalier-Boisvert et al., 2018), the agent receives as an observation a partial view of its surroundings (in our experiments it is set to two tiles to each side of the agent and four tiles in front) and a one-hot vector representing the agent's heading. The action space consists of 7 discrete actions but, in our setting, only the actions moving the agent forward and rotating it to the left or right have an effect. The episode starts with the agent at its start tile and facing its starting orientiation. The episode terminates successfully when the agent reaches the goal tile. It receives a reward between 0 and 1 based on the number of timesteps it took to get there. The episode will terminate without a reward if the agent steps on a lava tile, or when the maximum number of timesteps is reached.

Levels are parameterised as 2D grids representing the overall layout, with each tile type represented by an unique ID. Tiles can be classified as navigable (for example, moss or empty tiles) or non-navigable (for example, walls and lava, as stepping into lava terminates the episode). To be valid, a level must possess exactly one goal and start tile, and to be solvable there must exist a navigable path between the start and the goal location. We provide the color palette of tiles used in Figure 20.

We provide example levels of the CMDP in Figure 17. The CMDP level space corresponds to the subset of solvable levels in which moss and lava node placement is respectively positively and negatively correlated with their shortest path distance

to the goal.[5] Under partial observability, the optimal policy for this CMDP would leverage moss and lava locations as contextual cues, seeking regions with high moss density and avoiding regions with high lava density.



(a) Example training levels from X_{train} (b) Held-out levels

Figure 17: Sample levels from X_{train} and from held-out test set X_{test}. Wall tiles are rendered in gray, empty tiles in black, moss tiles in green and the goal tile in lime green. The agent is rendered as a blue or purple triangle, and is depicted at its start location. Each row corresponds to levels generated with a specific wave function collapse base pattern. Four different base patterns are used to generate X_{train} and X_{test} layouts. As in Procgen games, we find that the PPO agent exhibits a significant GenGap on held-out levels when being restricted to train on 200-500 levels. We therefore have X_{train} contain 512 levels and we have X_{test} contain 2048 levels for more accurate agent evaluation.

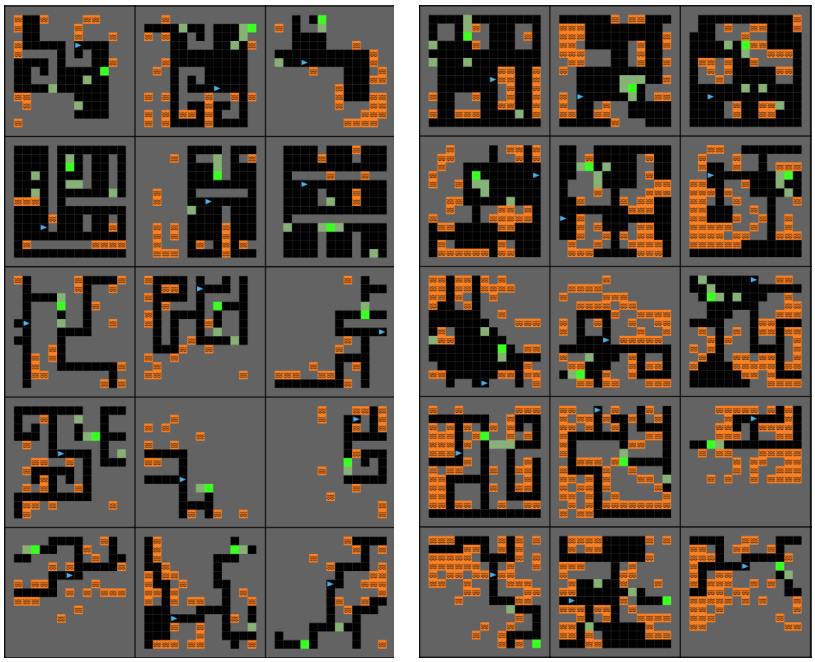
D.1. Generating highly structured levels

We use the wave function collapse (WFC) procedural generation algorithm (Gumin, 2016) to obtain highly structured but still diverse gridworld layouts. WFC gradually collapses a superposition of all possible level parameters into a layout respecting the constraints defined by an input pattern. By doing so, it is possible to generate a vast number of tasks from a small number of starting patterns. Given a suitable base pattern, WFC provides a high degree of structure and complexity in generated layouts, and it guarantees that both task structure and diversity scale with the gridworld dimensions. Our implementation supports generating layouts from 22 different base patterns, and supports the specification of custom user-defined patterns.

After generating a layout using WFC, we convert the navigable nodes of a layout into a graph, choose its largest connected component as the layout and convert any unreachable nodes to non-navigable nodes. We place the goal location at random and place the start at a node located at the median geodesic distance from the goal in the navigation graph. By doing so we ensure that the complexity of generated layouts is relatively consistent given a specific grid size and base pattern. Finally, we sample tiles according to parameterisable distributions defined over the navigable and non-navigable node sets.

In our experiments, the tile set consists of the { moss, empty, start, goal } tiles as the navigable set and the { wall, lava} tiles as the non-navigable set. We parameterise tile distributions such that moss tiles are more likely to be sampled on navigable nodes close to the goal, while lava tiles are more likely on non-navigable nodes far away from the goal.

[5]To measure the shortest path distance to goal of a non-navigable node, we first find the navigable node that is closest from it and measure its shortest path distance to the goal. We then add to it the number of tile separating this navigable node to the non-navigable node of interest. If there are multiple equally close navigable nodes, we select the navigable node with the smallest shortest path distance to the goal.



(a) Example levels from the first set of edge cases

(b) Example levels from the second set of edge cases

Figure 18: We generate 2 separate sets of edge cases, using 14 different base patterns not used to generate X_{train}. The moss density of the first set (a) is 2 to 5 times as small as levels found within X_{train}, making finding the goal using CMDP contextual cues more challenging. (b) is the same as (a) but with a lava tile density 2 to 5 times higher than X_{train} levels, which makes avoiding lava more challenging. Both sets contain the same number of levels (224), and they are combined when evaluating the agent on edge cases.

D.2. Controlling level complexity

We provide two options to vary the complexity of the level distribution. The first is to change the layout size. Due to partial observability this significantly increases their complexity, and makes levels from the "Hardcore" set depicted in Figure 6 challenging to solve for Human players. The second option is to change the sampling probability of moss and lava tiles. Reducing the fraction of moss to navigable tiles diminishes their usefulness as context cues. On the other hand, increasing the density of lava tiles increases the risk associated with selecting the wrong action during play. In our experiments we assess the agent's performance on edge-cases by defining level sets with a larger layout size (Figure 6), or with different moss and lava tile distributions (Figure 18).

E. Implementation details

E.1. Procgen

The Procgen Benchmark is a set of 16 diverse PCG environments that echoes the gameplay variety seen in the ALE benchmark (Bellemare et al., 2015). The game levels, determined by a random seed, can differ in visual design, navigational structure, and the starting locations of entities. All Procgen environments use a common discrete 15-dimensional action space and generate $64 \times 64 \times 3$ RGB observations. A detailed description of each of the 16 environments is provided in Cobbe et al. (2020). RL algorithms such as PPO reveal significant differences between test and training performance in all games, making Procgen a valuable tool for evaluating generalisation performance.

We conduct our experiment on the easy setting of Procgen, which employs 200 training levels and a budget of 25M training steps, and evaluate the agent's ZSG performance on the full range of levels, excluding the training levels.

We employ the same ResNet policy architecture and PPO hyperparameters (identical for all games) as (Cobbe et al., 2020), which we reference in Table 8. To compute $\mathrm{I}(L; b)$ and the scoring strategy for (S^{MI}) in our experiments, we model

$p_{\boldsymbol{\theta}}(i|b(o_t))$ as a linear classifier predicting the level identity from the output of the last shared network layer between the actor and critic. $p_{\boldsymbol{\theta}}$ is trained using trajectory rollouts collected by sampling uniformly from L. We ensure the training processes of the agent and the classifier remain independent from one-another by employing a separate optimiser, and by stopping the gradients from propagating through the agent's network.

E.2. Minigrid RL agent

We use the recurrent PPO agent and hyperparameters employed in (Parker-Holder et al., 2022) for all our experiments. The actor and critic share their initial layers. The first initial layer consists of a convolutional layer with 16 output channels and kernel size 3 processes the agent's view and a fully connected layer that processes its directional information. Their output is concatenated and fed to an LSTM layer with hidden size 256. The actor and critic heads each consist of two fully connected layers of size 32, the actor outputs a categorical distribution over action probabilities while the critic outputs a scalar. Weights are optimised using Adam and we employ the same hyperparameters in all experiments, reported in Table 8. Trajectories are collected via 36 worker threads, with each experiment conducted using a single GPU and 10 CPUs.

Following (Parker-Holder et al., 2022), DR and RPLR use domain randomisation as their standard level generation process, in which the start and goal locations, alongside a random number between 0 and 60 moss, wall or lava tiles are randomly placed. The level editing process of ACCEL and EL-DRED remains unchanged from (Parker-Holder et al., 2022), consisting of five steps. The first three steps may change a randomly selected tile to any of its counterparts, whereas the last two are reserved to replacing the start and goal locations if they had been removed in prior steps.

We train three different seeds for each baseline. We use the hyperparameters reported in (Parker-Holder et al., 2022) for the DR, RPLR and ACCEL methods and the hyperparameters reported in (Jiang et al., 2021b) for PLR, as an extensive hyperparameter search was conducted in a similarly sized Minigrid environment for each method. VAE-DRED employs the same hyperparameters as PLR for its level buffer, with some additional secondary sampling strategy hyperparameters introduced by VAE-DRED. We did not perform an hyperparameter search for VAE-DRED as we found that the initial values worked adequately. We report all hyperparameters in Table 8.

E.3. VAE architecture and pre-training procedure

We employ the $\beta-$VAE formulation proposed in (Higgins et al., 2017), and we parametrise the encoder as a Graph Convolutional Network (GCN), a generalisation of the Convolutional Neural Network (CNN) (Krizhevsky et al., 2012) to non Euclidian spaces. Our choice of a GCN architecture is not motivated by a desire of maximising the VAE's performance (in fact we expect a CNN or MLP encoder to work just as well in Minigrid). It is instead intended as a proof of concept for an architecture that could transfer to more complex simulators. Since the level parameter space $\mathbb{X}$ is simulator-specific, employing a graph as an input modality for our encoder makes our model architecture easier to transfer to different simulators and domains. Using a GCN, some of the inductive biases that would be internal in a traditional architecture can be defined through an external wrapper that encodes the environment parameter $\boldsymbol{x}$ into the graph $\mathcal{G}_{\boldsymbol{x}}$.

In Minigrid, we represent the gridworld levels as a grid graphs, each cell being an individual node. This effectively makes the GCN equivalent to a traditional CNN in this scenario. However a GCN provides ways of providing additional domain specific biases that a CNN lacks. For example, a domain expert with the notion that goal, lava and moss tiles are somehow correlated can add edges between these tiles in the graph.

We select the GIN architecture (Xu et al., 2019) for the GCN, which we connect to an MLP network that outputs latent distribution parameters $(\boldsymbol{\mu}_{\boldsymbol{z}}, \boldsymbol{\sigma}_{\boldsymbol{z}})$. The decoder is a fully connected network with three heads. The *layout* head outputs the parameters of categorical distributions for each grid cell, predicting the tile identity between [Empty, Moss, Lava, Wall]. The *start* and *goal* heads output the parameters of categorical distributions predicting the identity of the start and goal locations across grid cells, ensuring a single goal or start node get sampled in any given level.

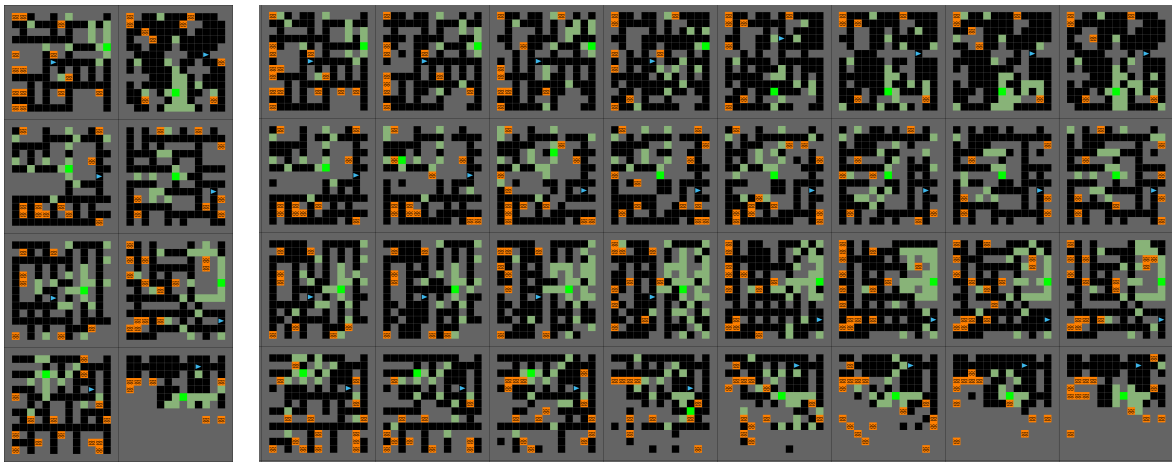
We pre-train the VAE for 200 epochs on X_{train}, using cross-validation for hyperparameter tuning, each run taking about 25 minutes on a GPU-equipped laptop. During training, we formulate the reconstruction loss as a weighted sum of the cross-entropy loss for each head.[6] At deployment, we guarantee *valid* layouts get generated (i.e. layouts containing a unique start and goal location, but not necessarily solvable) by generating level parameters sequentially. We first sample the layout, then we sample the start location, masking any non-navigable tile generated in the last step. We then add the start tile to

[6] To compute the cross-entropy loss of the layout head, we replace the start and goal nodes in the reconstruction targets by a uniform distribution across {moss, empty}.

our mask before sampling the goal location. In this way, we guarantee valid start and goal locations that will not override one-another. Note that our generative model may still generate *unsolvable* layouts, which do not have a passable path between start and goal locations, and therefore it must have learnt to generate solvable layouts in order to be useful.

We do not explicitly encourage the VAE to generate solvable levels, but we find that models with high ELBO (Equation (10)) on the validation set tend to also have a high generated layout solvability rate. Layouts reconstructed from X_{train} have over 80% solvability rate, while layouts generated via latent space interpolations have over 70% solvability rate. In practice, maximising the ELBO results in generated layouts sharing contextual semantics with X_{train} levels. This can be observed qualitatively in the VAE-generated layouts included in Figure 19, and quantitatively in Figures 13 and 14, where we report that contextually important semantics are transferred to the agent's training distribution.

To tune the VAE we conduct a random sweep over architectural parameters (number of layers, layer sizes), the β coefficient, individual decoder head reconstruction coefficients and the learning rate over a total budget of 100 runs. We select the configuration achieving the highest ELBO on the validation set. These hyperparameters are reported in Table 9.



(a) Parent levels sampled from X_{train}

(b) Levels generated by the VAE

Figure 19: Sample of levels generated by interpolating within the latent space of the VAE. On the right, each row represents an interpolation between the latent embeddings of a pair of base levels (on the left). The layouts generated are not always solvable, but, when they are solvable, they tend to retain semantics consistent with the CMDP.

Table 8: Hyperparameters used for Minigrid experiments. Hyperparameters shared between methods are only reported if they change from the method above.

Parameter	Procgen	MiniGrid
PPO		
γ	0.999	0.995
λ_{GAE}	0.95	0.95
PPO rollout length	256	256
PPO epochs	3	5
PPO minibatches per epoch	8	1
PPO clip range	0.2	0.2
PPO number of workers	64	32
Adam learning rate	5e-6	1e-4
Adam ϵ	1e-5	1e-5
PPO max gradient norm	0.5	0.5
PPO value clipping	yes	yes
return normalisation	yes	no
value loss coefficient	0.5	0.5
PLR		
Scoring function	ℓ_1-value loss	ℓ_1-value loss
Replay rate, p	1.0	1.0
Buffer size, K	200	512
Prioritisation,	rank	rank
Temperature,	0.1	0.1
Staleness coefficient, ρ	0.1	0.3
RPLR		
Scoring function,		positive value loss
Replay rate, p		0.5
Buffer size, K		4000
ACCEL		
Edit rate, q		1.0
Replay rate, p		0.8
Buffer size, K		4000
Edit method,		random
Levels edited,		easy
VAE-DRED		
Replay rate, p		1.0
Scoring function support,		dataset
Staleness support,		dataset
Secondary Scoring function,		ℓ_1-value loss
Secondary Scoring function support,		buffer
Secondary Temperature,		1.0
Mixing coefficient, η		linearly increased from 0 to 1

Table 9: Hyperparameters used for pre-training the VAE.

Parameter	
VAE	
β	0.0448
layout head reconstruction coefficient	0.04
start and goal heads reconstruction coefficients	0.013
number of variational samples	1
Adam learning rate	4e-4
Latent space dimension	1024
number of encoder GCN layers	4
encoder GCN layer dimension	12
number of encoder MLP layers (including bottleneck layer)	2
encoder MLP layers dimension	2048
encoder bottleneck layer dimension	256
number of decoder layers	3
decoder layers dimension	256

Figure 20: Color palette used for rendering minigrid layouts in this paper and their equivalent for Protanopia (Prot.), Deuteranopia (Deut.) and Tritanopia (Trit.) color blindness. We refer to each row in the main text as, in order: green (goal tiles), pale green (moss tiles), blue (agent), black (empty/floor tiles), grey (wall tiles) and orange (lava tiles).