Improving Regret Approximation for Unsupervised Dynamic Environment Generation

Harry Mead
University of Oxford

Bruno Lacerda
University of Oxford

Jakob Foerster
University of Oxford

Nick Hawes
University of Oxford

Abstract

Unsupervised Environment Design (UED) seeks to automatically generate training curricula for reinforcement learning (RL) agents, with the goal of improving generalisation and zero-shot performance. However, designing effective curricula remains a difficult problem, particularly in settings where small subsets of environment parameterisations result in significant increases in the complexity of the required policy. Current methods struggle with a difficult credit assignment problem and rely on regret approximations that fail to identify challenging levels, both of which are compounded as the size of the environment grows. We propose Dynamic Environment Generation for UED (DEGen) to enable a denser level generator reward signal, reducing the difficulty of credit assignment and allowing for UED to scale to larger environment sizes. We also introduce a new regret approximation, Maximised Negative Advantage (MNA), as a significantly improved metric to optimise for, that better identifies more challenging levels. We show empirically that MNA outperforms current regret approximations and when combined with DEGen, consistently outperforms existing methods, especially as the size of the environment grows. We have made all our code available here: `https://github.com/HarryMJMead/Dynamic-Environment-Generation-for-UED`.

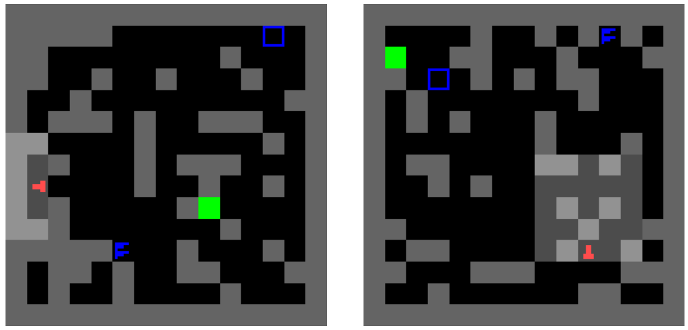
1 Introduction

Deep Reinforcement Learning (RL) has been effective in training highly-capable agents in a number of different challenging settings, such as in real-world robotics applications [1, 2, 25, 18], or games such as Go [29], Chess [30], Starcraft [31] and Dota [4]. However, these deep-RL agents tend to exhibit poor generalisation when transferred to tasks or environments with only small changes to those used to train on [33, 7].

In order to address this lack of robustness, domain-randomisation (DR), training over a diversity of environment parameterisations, has proven successful in a number of applications. However, DR relies on random parameterisations resulting in useful training examples, and in complex environments this may not be the case. Automated Curriculum Learning (ACL) [11, 23] methods aim to produce adaptive curricula for training that ensure the generation of useful training examples whilst maintaining a sufficiently diverse distribution over these environment parameterisations. These methods have shown success over naive domain-randomisation approaches [24, 20].

However, manually designing a suitable curriculum for learning may in itself be a challenge, whilst also limiting the capacity for open-ended learning [32]. Recent work has focused on Unsupervised Environment Design (UED) [9], which has emerged as a widely applicable curriculum design method as no prior environment knowledge is required. In the UED literature, each parameterisation of the environment is referred to as a level, and so UED frames the curriculum design problem as the interaction between a teacher agent designing levels and a student agent training on these levels. The majority of existing work focuses on maximising student regret [9, 17, 22, 6], as [9] shows that if the student and teacher reach a Nash equilibrium of a minimax regret game, the student must

39th Conference on Neural Information Processing Systems (NeurIPS 2025).



(a) No Key Required (b) Key Required

Figure 1: Examples of two possible randomly generated levels. In the first, the agent (red triangle) can simply navigate to the goal (green square), whereas in the second, it is required to first obtain the key (two blue triangles) in order to unlock the door (blue unfilled square) blocking the path to the goal

necessarily be able to solve all solvable environments. However, computing regret is intractable for many complex tasks, so these methods require *regret approximations*.

Generally UED methods can be categorised as either relying on a learnt level generator [9, 19, 3], or a curation process that selects and replays levels from a randomly generated set [16, 17, 22]. Existing UED methods have focused on environments such as minigrid [9, 6] or bipedal walker [22], where there is a relatively smooth transition in difficulty between levels. However, there are many environments where a small subset of paramaterisations may induce a step-change in difficulty for the student. For example, in the level shown in Figure 1(a), the addition of the door and key to the level generation have no effect on the difficulty of the level for the student agent. The key can simply be ignored and the door acts as any other wall, and so the agent is able to navigate to the goal directly. However, in Figure 1(b), the door blocks the agent's path to the goal, so it is necessary for the agent to first find the key before being able to unlock the door and reach the goal. In this example, these key-requiring mazes represent a very small subset of possible random levels, but they also represent the levels potentially most difficult to learn. Thus, for UED to be effective in environments such as these, it is necessary for methods to identify and train on this more challenging subset of levels.

Whilst replay-based methods are sufficient in small environments, the challenge of sampling and identifying more difficult subsets of levels is amplified as the size of the environment increases. Due to their reliance on random level generation, we show that replay methods fail in larger environments, and thus it is necessary to use learnt level generators. However, training a generator that generates a full level prior to student rollouts presents a challenging credit assignment task, given the long time horizon and sparse rewards. In order to address these challenges with learnt level generation, we propose Dynamic Environment Generation for Unsupervised Environment Design (DEGen). Our method involves dynamically generating the environment as the student agent explores the level, enabling a much denser teacher reward signal, and reducing the difficulty of credit assignment.

However, we show in this work that current regret approximations are insufficient, both for identifying these most difficult subsets of levels and for use in training level generators. We propose Maximised Negative Advantage (MNA) as a more effective regret approximation and show substantial empirical performance improvements over existing regret approximation metrics. Using MNA, we show that DEGen performs substantially better than existing generators that rely on full level generation upfront. We show that DEGen is capable of matching or exceeding the performance of existing replay methods in small environments, unlike previous learnt generators, but performs substantially better as the size of the environment increases. We also show that MNA consistently improves performance over current regret approximations for all UED methods.

Our contributions are:

- We introduce *Dyanamic Environment Generation for UED* (DEGen) as a new method of environment generation, showing performance improvements over existing learnt generators in small environments and significant performance improvements over all methods in larger environments.
- We introduce *Maximised Negative Advantage* (MNA) as a new regret approximation and show substantial improvements over existing metrics.

2 Background

2.1 Unsupervised Environment Design

Given a specific environment, we can model a level as a Partially Observable Markov Decision Process (POMDP). POMDPs can be defined by a tuple $\langle S, A, O, \mathcal{T}, \mathcal{I}, \mathcal{R}, \rho_0, \gamma \rangle$, where S, A and O are the set of states, actions and observations respectively, $\mathcal{T} : S \times A \to S$ is the transition function, mapping a state-action pair (s_t, a_t) to the subsequent state s_{t+1}, $\mathcal{I} : S \to O$ is the observation function that maps a given state to an observation, $\mathcal{R} : S \times A \to \mathbb{R}$ is the reward function, ρ_0 is the distribution over initial states, and γ is the discount factor.

In order to extend this formulation to the framework of UED, the Underspecified POMDP (UPOMDP) is introduced [9], defined by the tuple $\mathcal{M} = \langle S, A, O, \mathcal{T}^{\mathcal{M}}, \mathcal{I}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \rho_0^{\mathcal{M}}, \gamma \rangle$. The UPOMDP formulation introduces Θ, the set of all possible free environment parameters θ, for which a specific θ results in the environment configuration defined by the POMDP $\mathcal{M}_\theta$ with the transition, state and reward functions $\mathcal{T}^\theta, \mathcal{I}^\theta, \mathcal{R}^\theta$ and the initial state distribution ρ_0^θ.

Generally, the UED objective is to identify training levels that maximise the student's regret, given the current student policy π. The regret is defined as:

$$\text{Regret}(\pi, \theta) = -U(\pi, \theta) + U(\pi^*_\theta, \theta) \tag{1}$$

where π^*_θ is the optimal policy given θ, and $U(\pi, \theta) = \mathbb{E}_{\pi, \mathcal{M}_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$, or the expected discounted return of the policy π. As such, UED can be framed as a two player minimax regret game:

$$\min_{\pi \in \Pi} \max_{\theta \in \Theta} \text{Regret}(\pi, \theta). \tag{2}$$

By framing UED as this regret-based minimax game, if the environment satisfies the reward conditions outlined in [9], we can guarantee that if the student and teacher policies reach a Nash equilibrium, then the student policy must necessarily be capable of solving all solvable levels.

2.2 Existing UED Methods

Whilst the optimal objective shown in Equation 1 has robustness guarantees, in practise, it is infeasible for UED given π^*_θ is required. UED methods such as PAIRED [9] or CLUTR [3] introduce an additional antagonist agent, and regret is approximated as the difference between the performance of the antagonist and student policies. Both these methods rely on RL-trained teacher, where the teacher aims to maximise the performance difference between the antagonist and the student. However, these RL-trained teachers tend to struggle with maintaining diversity over training environments [16]. Some techniques have shown to improve performance, such as behaviour cloning between the antagonist and protagonist and the use of high entropy coefficients [19]. However these RL-based methods still tend to be outperformed by replay-based methods.

Rather than relying on a learnt generator, PLR [17], relies on maintaining a replay buffer of high-regret levels that have been sampled from a random generator. PLR alternates between sampling new random levels and replaying previously sampled levels. PLR relies on a score function to approximate regret, with the two most commonly used score functions being *Positive Value Loss* (PVL) and *Maximum Monte Carlo* (MaxMC) [16, 22].

PVL is defined as

$$\frac{1}{T} \sum_{t=0}^{T} \left(\max(0, \sum_{k=t}^{T} (\lambda\gamma)^{k-t} \delta_k) \right) \tag{3}$$

Algorithm 1 DEGen

Initialise: student policy π_{ϕ_1}, generator policy Λ_{ϕ_2}
while not converged **do**
  // Sample N trajectories
  for $n \in 1 : N$ **do**
    Initialise empty level
    // Take T student steps
    for $t_s \in 1 : T$ **do**
      // Generate partial level
      Sample Λ actions to generate section of level that has been observed but not generated
      // Take student action
      Sample π action
    end for
    compute score using student trajectory τ_s
    assign reward to generator trajectory τ_g
  end for
  Update ϕ_1 according to sampled student trajectories
  Update ϕ_2 according to sampled generator trajectories
end while

where γ is the discount factor, λ is from the Generalised Advantage Estimator [27] and δ_t is the 1-step TD-error at timestep t. We can view PVL as approximating regret as the average advantage, but with the advantage clipped at 0. As such, maximising PVL can be seen as effectively maximising states where the student does better than expected, which intuitively appears to be mismatched with the regret objective. Despite this, empirically, PVL has been shown to be effective.

MaxMC approximates regret using the maximum achieved return (R_{max}) on a given level, and is defined as

$$\frac{1}{T}\sum_{t=0}^{T}\left(R_{max} - \hat{V}(s_t)\right) \tag{4}$$

where $\hat{V}(s_t)$ is the learnt value function approximation for the value of the current policy at state s_t. MaxMC appears a more intuitive approximation for regret than PVL. However by relying on a Monte Carlo approximation for regret, MaxMC requires a sufficient number of trials in a level such that R_{max} is a good approximation for the optimal return. Additionally, MaxMC can only be used in environments where reward is obtained at the final step, as R_{max} is dependent on the full episode reward.

Whilst PLR has been shown to be effective in a number of domains, relying on random level generation necessarily means that no insight is gained from past levels to influence future levels generation. ACCEL [22] addresses this by augmenting PLR such that new levels are generated by mutating existing levels previously in the replay buffer. This evolutionary approach enables some capacity for learning from previously identified high-regret levels, but still relies on random mutations for new level generation.

An alternate approach for UED is Sampling for Learnability (SFL) [26]. Similarly to PLR, SFL relies on sampling a set of randomly generated levels and selecting those with the highest scores for training. However, rather than this score metric approximating regret, SFL aims to train on levels with high learnability, defined as $p(1-p)$, where p is the success rate of the current policy on the sampled level.

3 Dynamic Environment Generation

Existing work [16, 22, 19] has shown that replay-based UED methods generally outperform methods relying on a learnt generator. This can be attributed to a number of factors, but primarily, level generation presents difficulties for learning. The level generation learning problem has a long time horizon and sparse rewards, and so credit assignment is challenging. Current UED have focused on relatively small environments where it is feasible to sample useful training levels from random level generation. However, as the size of the environment grows, and especially with environments with features that can add complexity, it becomes more difficult to sample useful levels. Therefore, it becomes necessary to use a learnt generator instead.

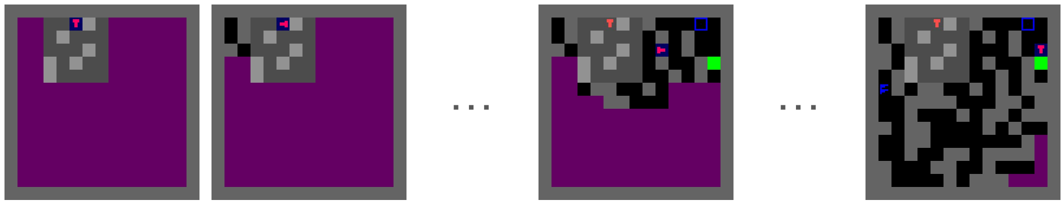


Figure 2: Illustration of how DEGen generates the level as the student agent explores the level, with purple colouring indicating areas that are yet to be observed.

In order to address this, whilst also reducing the challenges inherent to RL-based learnt generators, we propose Dynamic Environment Generation (DEGen), outlined in Algorithm 1. Rather than simply generating the entire environment level initially, we exploit the partial observability of the student agent and instead only generate those parts of the level the student observes, as illustrated in Figure 2. This allows for a much denser reward signal, reducing the difficulty of credit assignment.

If we consider the PVL and MaxMC regret approximations shown in Equations 3 and 4 respectively, both can be written in the form

$$\text{Regret} \approx \frac{1}{T}\sum_{t=0}^{T} G_t, \tag{5}$$

where we approximate regret as the mean of some value G_t across the student trajectory. If the entire level is generated initially, this approximation must simply be assigned as the reward for the last step in the level generation trajectory, resulting in a very sparse reward signal for the generator. However, if the level is generated as the student trajectory unrolls, we are able to assign a much denser generator reward. If we have the function $t_s = \mathcal{T}(t_g)$ that maps the generator timestep t_g to the student timestep t, we can instead assign the generator reward (r_g) at timestep t_g as

$$r_{t_g} = \frac{1}{T}\sum_{t=\mathcal{T}(t_g)}^{\mathcal{T}(t_g+1)-1} G_t. \tag{6}$$

This increased reward density reduces some of the difficulty of the credit assignment challenge for training the level generator. Additionally, DEGen reduces noise in the credit assignment by only generating parts of the level the student observers. If the entire level is generated initially, it is likely that some parts of the level will never be observed by the student, and so have no effect on the score of the level. As such, these actions only serve to add additional noise to the credit assignment task, an issue which is negated by DEGen.

Another issue present in existing RL-based generators is a lack of diversity in generated levels. Previous work [19] has shown that a high entropy coefficient can reduce this issue, however we still observed reduced level diversity with existing RL-based generators. As the student policy π is stochastic, by generating the level based on where the student explores, we introduce an greater degree of stochasticity in the level generation, which increases the diversity of generated levels. We also found that introducing some additional randomness in level generation - specifically randomly initialising the starting location of the student agent - improved training level diversity, and zero shot agent performance.

4 Maximised Negative Advantage

Whilst the PVL and MaxMC regret approximations have shown success in a number of domain, there are flaws with both metrics in relation to generating challenging levels. For PVL, high positive advantage will result in a high PVL score, whereas generally, more challenging levels would tend to be more difficult than expected, and so more likely to result in high negative advantage. For MaxMC, high scores require at least one high return rollout, and again, for the most challenging levels, this is unlikely to occur. Whilst PVL and MaxMC have shown success with use in replay-based methods, we observed poor performance when these metrics were directly optimised for, shown in Appendix C.2. We instead propose Maximised Negative Advantage (MNA) as a more suitable metric.

Consider the policy π with the value-function $V(s)$. We are aiming for an approximation of Equation 1, requiring an approximation for both $U(\pi^*_\theta, \theta)$ and $U(\pi, \theta)$. If we assume a deterministic environment, given a trajectory of length T, if $V(s)$ is the true value function, we can lower bound $U(\pi^*_\theta, \theta)$

$$U(\pi^*_\theta, \theta) \geq \max \begin{pmatrix} V(s_0), \\ \gamma V(s_1) + r(s_0, a_0), \\ \vdots \\ \gamma^T V(s_T) + \sum_{k=0}^{T-1} \gamma^k r(s_k, a_k) \end{pmatrix}. \tag{7}$$

We label this maximum over value functions

$$V_n^{\max}(s_t) = \max \begin{pmatrix} V(s_t), \\ \vdots \\ \gamma^n V(s_{t+n}) + \sum_{k=t}^{t+n-1} \gamma^{k-t} r(s_k, a_k) \end{pmatrix}. \tag{8}$$

Given this, as the true value function $V(s_0)$ gives us the expected performance of the current policy, and the maximum over value functions $V_T^{\max}(s_0)$ lower bounds the performance of the optimal policy, we can lower bound the regret as

$$\text{Regret} \geq -V(s_0) + V_T^{\max}(s_0). \tag{9}$$

However, in practise, the exact value function $V(s)$ will generally be unknown, and instead must be approximated with a learnt value function $\hat{V}(s)$. As this learnt value function may overestimate the value of the state, the inequality in Equation 7 does not necessarily hold when $V(s)$ is replaced with $\hat{V}(s)$. Additionally, the $\hat{V}(s_0)$ approximation for $U(\pi, \theta)$ will be biased, being a learnt value approximation. We can reduce the likelihood of $\hat{V}_T^{\max}(s_0)$ exceeding $U(\pi^*_\theta, \theta)$ by instead using the approximation $\hat{V}_n^{\max}(s_0)$, where $n < T$, although this instead results in a potentially overly conservative approximation. Similarly, we can reduce the bias of our approximation for $U(\pi, \theta)$ by instead using the approximation $\gamma^n V(s_n) + \sum_{k=t}^{n-1} \gamma^k r(s_k, a_k)$, however this then introduces greater variance. We therefore define the n-step regret approximation at timestep t as

$$\hat{G}_t^{(n)} = -\Big(\gamma^n V(s_{t+n}) + \sum_{k=t}^{t+n-1} \gamma^{k-t} r(s_k, a_k)\Big) + \hat{V}_n^{\max}(s_t) \tag{10}$$

and we note the similarity between this regret approximation and the negative n-step advantage estimation [27]. In order to balance both the bias and variance of the $U(\pi, \theta)$ approximation, and the conservativeness of the $U(\pi^*_\theta, \theta)$ approximation, in line with the Generalised Advantage Estimator [27], we introduce the regret approximation

$$\hat{G}_t^\lambda = (1 - \lambda) \sum_{n=0}^{\infty} \lambda^n \hat{G}_t^{(n)}. \tag{11}$$

Empirically, we find that rather than just approximating regret at the first state, using the mean regret approximation was a more effective metric

$$\frac{1}{T} \sum_{t=0}^{T} \hat{G}_t^\lambda. \tag{12}$$

We show empirically that this regret approximation is much suitable optimisation metric for learnt generators, whilst also showing improved performance when used in replay-based methods.

4.1 Solvability

Whilst the metric in Equation 12 does allow for more challenging levels to be sampled than existing metrics, the reliance on the learnt value function $\hat{V}(s)$ to determine maximum possible performance does present issues with ensuring level solvability. If the environment satisfies the reward conditions [9] that ensure the teacher-student regret Nash equilibrium results in an student policy capable of solving all possible solvable levels, then regret is maximised by generating solvable levels. Therefore, a good regret approximation metric should not result in high scores for unsolvable levels. Both PVL and MaxMC implicitly score low for unsolvable levels. For the Nash equilibrium result to hold, the reward conditions [9] necessitate that the maximum achievable return for an unsolvable level F_{max} must not exceed the minimum return achievable in a solved level S_{min}. In the case of MaxMC, a higher R_{max} can be achieved if the level is solvable, and so solvable levels will generally score higher. For PVL, higher returns will tend to result in higher advantage, and therefore a higher score, so solvable levels that can achieve higher returns will score high.

The issue with this implicit bias towards solvable levels is that in practice, this manifests as a bias towards levels with a high success rate [26], i.e. the current policy solves the level with a high probability. Therefore, these metrics generally do not produce sufficiently challenging levels for training. SFL [26] has shown that training using levels with approximately 50% success rate results in strong training performance. However, determining the exact success rate for a given level requires substantially more environment rollouts than necessary for metrics such as PVL or MaxMC.

While MNA is capable of identifying challenging levels with significantly fewer rollouts than directly scoring based on success rate, over-approximations of state value $\hat{V}(s)$ may result in high scores for unsolvable levels. In order to compensate for this, we introduce an explicit penalisation for unsolvable levels. As it is often intractable to determine exact solvability of levels in complex environments, we instead introduce approximate unsolvablilty, where we define a level as approximately unsolvable if it has never been solved, e.g. has a success rate of 0%. Therefore, if a level is approximately unsolvable, we set the score for the level to zero. As such, our final proposed regret approximation is

$$\text{MNA} = \left(\frac{1}{T} \sum_{t=0}^{T} \hat{G}_t^{\lambda} \right) \cdot \hat{C} \tag{13}$$

where $\hat{C}$ is 0 if the level is approximately unsolvable and 1 otherwise.

5 Experimental Setup

For this work, we examine the standard minigrid environment used in previous UED work [9, 16, 22], as well as evaluating UED performance on the modified minigrid with the addition of a key and locked door. In line with exisiting UED work, our main evaluation metric is zero-shot performance on a set of hand-designed test levels. For the standard minigrid, we use the set of 8 test levels used in previous work [8, 26]. For the modified key minigrid, we modify this set of levels so as to require the agent to unlock the door to reach the goal. Previous work has only examined minigrid when students are trained using 13x13 levels, but in order to scale UED to larger environments, we need to ensure that the student is still capable of learning complex skills, such as unlocking a door, even when trained in larger environments. To assess the student's ability to solve levels that require the door to be unlocked, we evaluate student performance on the existing key minigrid test levels but when trained on levels that are 17x17 and 21x21.

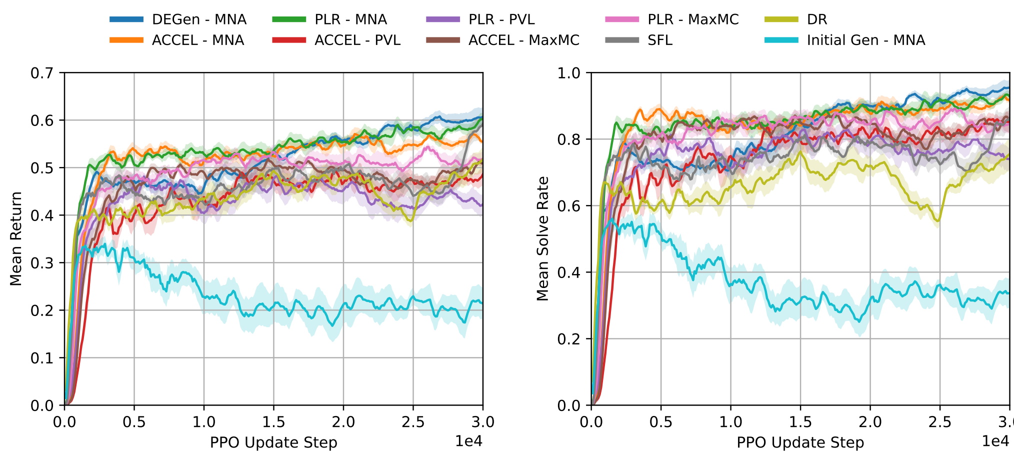
Baselines: In order to assess the effectiveness of both MNA and DEGen, we compare a number of different existing level generation methods and a number of regret approximation metrics. For regret approximation metrics, we compare MNA to the existing metrics, MaxMC and PVL. For assessing these metrics, we use the existing UED methods PLR [16] and ACCEL [22]. We include Domain Randomisation (DR) to show the relative performance of UED compared to a naive, random approach. Additionally, we include SFL [26] for a non-regret based approach. For the standard minigrid environment, we also include the Initial Gen baseline, corresponding to an RL-trained teacher that generates the full environment prior to student rollouts, however we show that this performs substantially worse than all other methods so do not include it in the key minigrid domain. All student agents are trained using PPO [28], as well as the teacher agents used in DEGen and Initial

Gen. Detailed training hyperparameters for all domains and UED methods are found in Appendix A.2.

6 Results

6.1 Minigrid

Figure 3 illustrates the performance of MNA and DEGen compared to existing baselines. From these plots, it is clear that Intial Gen performs substantially worse than all other methods, whereas DEGen performs comparably to existing replay-based methods. This substantial performance deficit can likely be attributed to a lack of diversity in the generated levels, see Appendix C.5. We see a far greater diversity in the levels generated via DEGen, and this, along with the reduced credit assignment challenge, allows for DEGen to substantially outperform Initial Gen, despite the former also relying on an RL-trained teacher. We also see that for both PLR and ACCEL, MNA outperforms the existing MaxMC and PVL regret approximations. However, Figure 3 also shows that in this setting, early in training, DEGen is outperformed by replay-based methods using MNA. This suggests that, whilst the final DEGen performance is equal or greater than the performance of replay-based methods, the additional challenge of learning the generator does impact initial performance.



(a) Mean Return (b) Mean Solve Rate

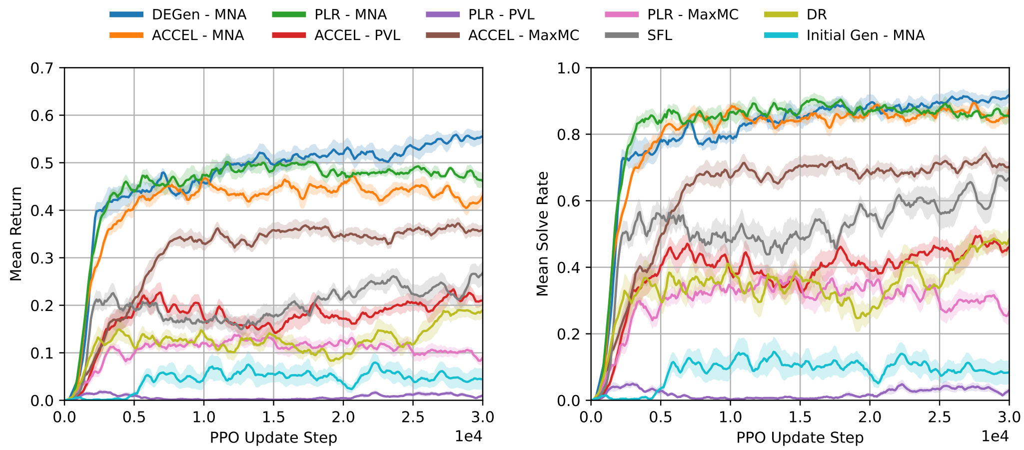
Figure 3: Minigrid zero-shot performance on hand-designed test set, showing mean and standard error across 8 runs.

6.2 Key Minigrid

Figure 4 compares the performance of the various tested UED methods in the key minigrid domain. Due to the increased challenge of level generation with the addition of a key and locked door, there is far more variance in the relative performance of each of the methods. In this domain, we see DEGen outperform all existing baselines, as well as the MNA-based replay methods. We see that PLR using either MaxMC or PVL performs extremely poorly, whereas PLR using MNA, which is much more capable of identifying challenging levels, is the best performing baseline. This highlights the improved regret approximation of MNA compared to existing metrics. We also note that SFL performs poorly in this key-minigrid setting. Learnability only considers the final outcome of an episode, rather than the full student trajectory, and this result suggests that this may be insufficient in domains where additional environment features result in a more complex subset of levels.

6.3 Increased Environment Size for Key Minigrid

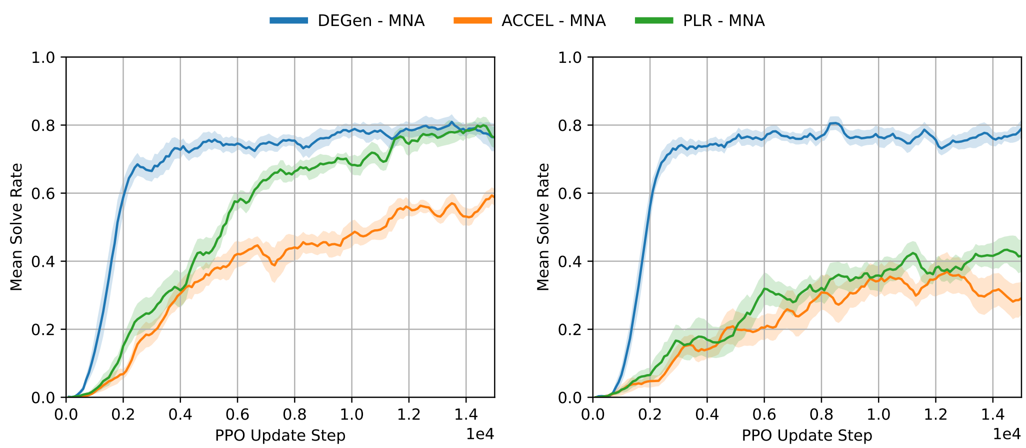
Up to this point, we have examined only the relatively small 13x13 environment setting. However, as outlined above, with the aim of scaling UED to larger, more complex environments, we examine the



(a) Mean Return

(b) Mean Solve Rate

Figure 4: Minigrid with key and locked door zero-shot performance on hand-designed test set, trained on 13x13 training levels, showing mean and standard error across 8 runs.



(a) 17x17

(b) 21x21

Figure 5: Minigrid with key and locked door zero-shot performance on hand-designed test set, trained on larger training levels, showing mean and standard error across 8 runs.

performance of current UED methods when simply scaling up a version of minigrid to sizes larger than 13x13. Figure 5 shows the performance of DEGen, PLR and ACCEL using MNA when training levels are 17x17 and 21x21. Whilst there is a performance drop compared to the 13x13 levels, as the size of the environment grows, DEGen substantially outperforms existing methods. This result highlights that, with any domains with additional environment features such as the key and door that can potentially add additional complexity to policy learning, as the environment size grows, it becomes substantially harder to sample useful levels where these features are necessary. Therefore it is necessary to use a trained generator and DEGen is able to overcome the credit assignment challenges present in previous methods for training learnt generators.

7 Discussion

It is clear that from these results that DEGen does substantially outperform existing baselines in both the minigrid, and key minigrid environments, and this performance improvement is increased as the environment size increases. Scaling environment size is likely a necessity if future work aims to move UED from the current set of toy small-scale environments used, to more useful real-world domains, and this work shows that replay-based methods perform poorly with relatively small increases to environment size. We have demonstrated that DEGen is an effective method of addressing these issues, both compared to generating the full level prior to student rollouts, and replay-based methods.

8 Limitations

Whilst the results we have presented in this paper do show strong performance from DEGen compared to existing baselines, the domains presented in this paper present relatively simple mappings between the level representation and the agent's current observation. In more complex domains, such as 3D Games [15, 10, 13], or real-world robotics applications [21, 34, 14], it will be more complex to determine how specific environment parameters affect what the agent is currently observing. In order for UED to bridge the gap from the current set of game domains to real-world applications, it would be necessary for DEGen or DEGen-like methods to address this limitation. We believe that World Models [12] represent a promising direction for future research to address this. In its current form, DEGen relies on both the agent and the generator interacting with a fixed environment, where the environment has an explicit mapping between the agent's observation and the level parameters, and the generator is only able to generate the level where the agent has observed. However, a world model guided by a regret approximation such as MNA would represent a generator that could directly generate observations for the agent. Rather than relying on explicit mapping between level and observation, this mapping could be learnt with environment data when training the world model. Whilst the training of a world model would add additional computational cost to the training process, it would enable DEGen methodology to be applied to substantially more complex environments. With the advent of highly general world models such as the Genie series of world models [5], this could represent a path to training highly general policies that are effective in a wide variety of applications.

Additionally, in line with previous work [16, 22, 6], we have compared the relative performance of UED methods based on the number of student PPO update steps. However, training the DEGen teacher agent adds a high computational cost to the training loop, and so training using DEGen takes approximately four times as long as training using methods such as PLR and ACCEL. Full details on compute time and experiment specification can be found in Appendix A. Therefore, replay-based methods may be preferable for small environment sizes where similar performance is achieved. However again, it is clear than as environment size grows, the maximum performance achieved by DEGen exceeds replay-based methods, and so the additional time cost is justified, given the performance gains.

9 Conclusion

In this paper, we introduce a new level generation method, Dynamic Environment Generation for UED, and a new regret approximation metric, Maximised Negative Advantage. We outline how current UED methods fail as training environment size increases, and show that DEGen is capable of mitigating the issues associated both with these larger environments, and with RL-based level generation. We show that the use of MNA enables DEGen to outperform existing baselines, whilst also showing that the use of MNA consistently improves the performance of existing UED methods. These performance improvements are most evident in the more complex key minigird domain. We believe there is significant potential for future UED research to address larger and more complex environments, and that approaches based on MNA and DEGen provide a promising foundation for this advancement.

Acknowledgements

This work was supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems [EP/S024050/1]. Lacerda and Hawes have received EPSRC funding via the "From Sensing to Collaboration" programme grant [EP/V000748/1].

References

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[3] Abdus Salam Azad, Izzeddin Gur, Jasper Emhoff, Nathaniel Alexis, Aleksandra Faust, Pieter Abbeel, and Ion Stoica. Clutr: Curriculum learning via unsupervised task representation learning. In *International Conference on Machine Learning*, pages 1361–1395. PMLR, 2023.

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

[5] Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

[6] Hojun Chung, Junseo Lee, Minsoo Kim, Dohyeong Kim, and Songhwai Oh. Adversarial environment design via regret-guided diffusion models. *arXiv preprint arXiv:2410.19715*, 2024.

[7] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International conference on machine learning*, pages 1282–1289. PMLR, 2019.

[8] Samuel Coward, Michael Beukman, and Jakob Foerster. Jaxued: A simple and useable ued library in jax. *arXiv preprint arXiv:2403.13091*, 2024.

[9] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

[10] Raihana Ferdous, Fitsum Kifetew, Davide Prandi, and Angelo Susi. Towards agent-based testing of 3d games using reinforcement learning. In *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*, pages 1–8, 2022.

[11] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR, 2018.

[12] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3), 2018.

[13] Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models. *arXiv preprint arXiv:2509.24527*, 2025.

[14] Dong Han, Beni Mulyana, Vladimir Stankovic, and Samuel Cheng. A survey on deep reinforcement learning algorithms for robotic manipulation. *Sensors*, 23(7):3762, 2023.

[15] Jack Harmer, Linus Gisslén, Jorge del Val, Henrik Holst, Joakim Bergdahl, Tom Olsson, Kristoffer Sjöö, and Magnus Nordin. Imitation learning with concurrent actions in 3d games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

[16] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021.

[17] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.

[18] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

[19] Ishita Mediratta, Minqi Jiang, Jack Parker-Holder, Michael Dennis, Eugene Vinitsky, and Tim Rocktäschel. Stabilizing unsupervised environment design with a learned adversary. In *Conference on Lifelong Learning Agents*, pages 270–291. PMLR, 2023.

[20] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020.

[21] Hai Nguyen and Hung La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE international conference on robotic computing (IRC)*, pages 590–595. IEEE, 2019.

[22] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR, 2022.

[23] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pages 835–853. PMLR, 2020.

[24] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

[25] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[26] Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. *arXiv preprint arXiv:2408.15099*, 2024.

[27] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

[29] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[30] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[31] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

[32] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

[33] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[34] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021.

A Detailed Experimental Setup

A.1 Environment Details

Minigrid

We use the standard minigrid implementation from exisiting UED work [9, 16, 22, 26, 6]. Examples of levels are shown in Figure 6. The agent, depicted in red, observes a 5x5 square in front of it. The agent receives reward when reaching the goal, with a higher reward achieved when reaching the goal in fewer steps. At each step, the agent can either move *forward*, turn *left* or turn *right*.

Key Minigrid

The key minigrid implementation is identical to the standard minigrid level, except for the the addition of a key and locked door. The agent has an additional *use* action. The agent will pick up the key if it reaches the grid square the key is on. In order to unlock the door, the agent must select the *use* action when it has the key and the door is directly ahead of the agent.

DEGen Teacher

The teacher's observations are an augmented version of the student agent. Similarly to the student, the teacher observes a 5x5 square in front of the student. However, this is augmented by an overlayed 5x5 square that indicates which squares have yet to be generated.

At each step, the teacher is able to fill any ungenerated grid square the student is currently observing. The teacher must fill all ungenerated cells currently observed before the student is able to take another action. The teacher action space is split into two actions (a_1, a_2) where a_1 selects which cell will be filled, and a_2 selects what the cell will be filled with, e.g. *wall*, *empty*, *goal*, *key*, *door*. We use action masking to ensure that only previously ungenerated cells are filled, as well as enforce that only one *goal*, *key* and *door* can be placed for each level.

As the level is rolled out, at each step, either the teacher or the student will act. If all observed squares have been generated, the student will act, otherwise the teacher will act. As both the student and teacher have partial observability, policy networks include an LSTM layer, so actions are conditioned on all previous observations. The teacher is conditioned on all previous environment observations, whereas the student is only conditioned on fully-generated observations.

A.2 Hyperparameters

Full code and instructions on how to run can be found at `https://github.com/HarryMJMead/Dynamic-Environment-Generation-for-UED`.
All existing methods were trained using implementations based on JaxUED [8], available at `https://github.com/DramaCow/jaxued`, and SFL [26], available at `https://github.com/amacrutherford/sampling-for-learnability`. Learning hyperparameters are shown in Table 1 and the replay UED hyperparameters are shown in Table 2.

Table 1: Learning Hyperparameters.

Parameter	Minigrid	Key Minigrid 13x13	17x17 and 21x21
Student PPO			
Number of Updates	30000		15000
γ	0.995		
λ_{GAE}	0.95		
PPO number of steps	512		
PPO epochs	4		
PPO minibatches per epoch	4		
PPO clip range	0.04		
PPO # parallel environments	256		
Adam learning rate	5e-4		2.4e-4
Anneal LR	yes		no
Adam ϵ	1e-5		
PPO max gradient norm	0.5		
PPO value clipping	yes		
value loss coefficient	0.5		
entropy coefficient	1e-3		
Hidden dimension size	256		
Teacher PPO			
γ	0.998		0.99
λ_{GAE}	0.95		
PPO epochs	4		
PPO minibatches per epoch	4		
PPO clip range	0.2		
Adam learning rate	1e-3		
Anneal LR	yes		no
Adam ϵ	1e-5		
PPO max gradient norm	0.5		
PPO value clipping	yes		
value loss coefficient	0.5		
entropy coefficient	5e-2		
Hidden dimension size	256		
Num Teacher Steps *(Initial Gen)*	60		

Table 2: UED Hyperparameters.

Parameter	Minigrid	Key Minigrid
PLR		
Replay rate, p	0.5	
Buffer size, K	8000	
Prioritisation	Rank	
Temperature, β	1.0	
staleness coefficient	0.3	
ACCEL		
Number of Edits	20	
Buffer size, K	8000	
Prioritisation	Rank	
Temperature, β	1.0	
SFL		
Batch Size N	25000	
Rollout Length L	20000	
Update Period T	100	
Buffer Size K	1000	
Sample Ratio ρ	0.5	

A.3 Compute Details

For all experiments, each run was on 1 Nvidia A40. We show the mean compute time for both domains and each UED method in Table 3.

Table 3: Compute Time.

	Compute Time (hh:mm)	
Method	**Minigrid**	**Key Minigrid**
DR	11:16 ± 00:02	12:01 ± 00:02
Initial Gen	13:39 ± 00:01	13:49 ± 00:02
PAIRED	23:32 ± 00:02	23:33 ± 00:01
SFL	10:53 ± 00:01	10:48 ± 00:02
PLR	06:53 ± 00:02	06:53 ± 00:01
ACCEL	05:43 ± 00:02	05:45 ± 00:01
DEGen	25:49 ± 00:00	25:53 ± 00:01

A.4 Zero-shot Transfer Levels

Figures 6 and 7 show the hand designed levels used for evaluating zero-shot performance. The minigrid levels were taken directly from JaxUED [8]. The key minigrid levels have been modified so that the student is required to unlock the door to reach the goal. Note that we have chosen to include the FourRooms_Key levels rather than modified versions of the labyrinth levels, as the key would trivially be on the path to the goal for these labyrinth levels.

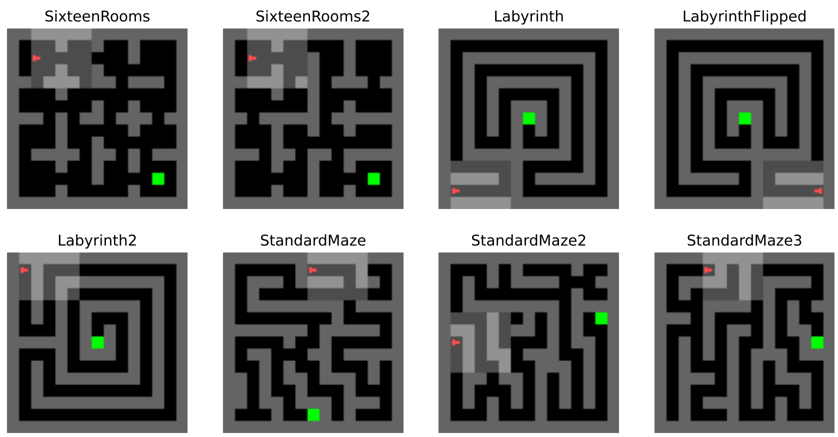


Figure 6: Hand designed evaluation levels for minigird

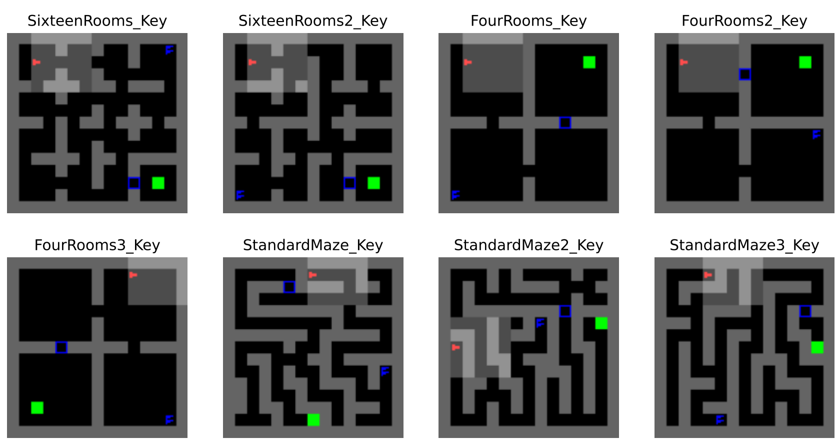


Figure 7: Hand designed evaluation levels for key minigird

A.5 Tabular Results

Minigrid Results

Table 4: Minigrid Solve Rate (1)

Level	Initial Gen - MNA	DR	SFL	PLR - MaxMC	ACCEL - MaxMC
SixteenRooms	0.93 ± 0.06	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**
SixteenRooms2	0.48 ± 0.14	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**
Labyrinth	0.04 ± 0.04	0.51 ± 0.16	0.60 ± 0.13	0.45 ± 0.12	0.69 ± 0.13
LabyrinthFlipped	0.11 ± 0.10	0.38 ± 0.15	0.68 ± 0.13	0.64 ± 0.12	0.50 ± 0.13
Labyrinth2	0.05 ± 0.04	0.40 ± 0.14	0.83 ± 0.12	0.75 ± 0.07	0.81 ± 0.09
StandardMaze	0.28 ± 0.11	0.99 ± 0.01	**1.00 ± 0.00**	0.98 ± 0.02	0.90 ± 0.05
StandardMaze2	0.30 ± 0.10	0.85 ± 0.06	0.95 ± 0.05	0.84 ± 0.12	**1.00 ± 0.00**
StandardMaze3	0.56 ± 0.14	0.93 ± 0.04	**1.00 ± 0.00**	0.98 ± 0.02	**1.00 ± 0.00**
Mean	0.34 ± 0.05	0.76 ± 0.05	0.88 ± 0.04	0.83 ± 0.03	0.86 ± 0.02

Table 5: Minigrid Solve Rate (2)

Level	PLR - PVL	ACCEL - PVL	PLR - MNA	ACCEL - MNA	DEGen - MNA
SixteenRooms	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**
SixteenRooms2	0.95 ± 0.04	0.88 ± 0.05	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**
Labyrinth	0.71 ± 0.09	0.86 ± 0.04	0.88 ± 0.06	0.68 ± 0.09	**0.98 ± 0.02**
LabyrinthFlipped	0.63 ± 0.11	0.78 ± 0.07	0.73 ± 0.11	0.73 ± 0.08	**0.91 ± 0.09**
Labyrinth2	0.51 ± 0.07	0.73 ± 0.08	0.93 ± 0.06	**0.95 ± 0.02**	0.79 ± 0.12
StandardMaze	0.63 ± 0.09	0.84 ± 0.06	0.98 ± 0.02	**1.00 ± 0.00**	**1.00 ± 0.00**
StandardMaze2	0.63 ± 0.10	0.88 ± 0.07	0.86 ± 0.08	0.99 ± 0.01	**1.00 ± 0.00**
StandardMaze3	0.91 ± 0.05	0.99 ± 0.01	**1.00 ± 0.00**	0.98 ± 0.02	**1.00 ± 0.00**
Mean	0.75 ± 0.03	0.87 ± 0.03	0.92 ± 0.02	0.91 ± 0.01	**0.96 ± 0.03**

Key Minigrid 13x13 Results

Table 6: Key Minigrid 13x13 Solve Rate (1)

Level	Initial Gen - MNA	DR	SFL	PLR - MaxMC	ACCEL - MaxMC
SixteenRooms_Key	0.18 ± 0.12	0.59 ± 0.12	0.58 ± 0.12	0.41 ± 0.11	0.83 ± 0.04
SixteenRooms2_Key	0.12 ± 0.12	0.55 ± 0.13	0.83 ± 0.10	0.19 ± 0.05	0.83 ± 0.08
FourRooms_Key	0.03 ± 0.02	0.55 ± 0.13	**0.98 ± 0.02**	0.10 ± 0.08	0.96 ± 0.02
FourRooms2_Key	0.05 ± 0.04	0.88 ± 0.06	0.93 ± 0.07	0.79 ± 0.10	0.98 ± 0.02
FourRooms3_Key	0.29 ± 0.14	0.51 ± 0.14	0.85 ± 0.06	0.10 ± 0.06	0.95 ± 0.05
StandardMaze_Key	0.01 ± 0.01	0.09 ± 0.04	0.35 ± 0.12	0.24 ± 0.07	0.20 ± 0.10
StandardMaze2_Key	0.00 ± 0.00	0.61 ± 0.13	0.60 ± 0.14	0.40 ± 0.11	0.26 ± 0.07
StandardMaze3_Key	0.00 ± 0.00	0.36 ± 0.14	0.30 ± 0.14	0.09 ± 0.06	0.75 ± 0.05
Mean	0.08 ± 0.04	0.52 ± 0.05	0.68 ± 0.04	0.29 ± 0.03	0.72 ± 0.02

Table 7: Key Minigrid 13x13 Solve Rate (2)

Level	PLR - PVL	ACCEL - PVL	PLR - MNA	ACCEL - MNA	DEGen - MNA
SixteenRooms_Key	0.15 ± 0.10	0.55 ± 0.04	0.95 ± 0.03	0.99 ± 0.01	**1.00 ± 0.00**
SixteenRooms2_Key	0.00 ± 0.00	0.63 ± 0.10	0.90 ± 0.06	0.88 ± 0.10	**1.00 ± 0.00**
FourRooms_Key	0.00 ± 0.00	0.60 ± 0.12	0.95 ± 0.04	**0.98 ± 0.02**	0.94 ± 0.05
FourRooms2_Key	0.06 ± 0.03	0.80 ± 0.06	**1.00 ± 0.00**	0.99 ± 0.01	**1.00 ± 0.00**
FourRooms3_Key	0.00 ± 0.00	0.66 ± 0.09	0.95 ± 0.03	0.98 ± 0.02	**1.00 ± 0.00**
StandardMaze_Key	0.00 ± 0.00	0.00 ± 0.00	0.46 ± 0.10	0.85 ± 0.08	**0.93 ± 0.06**
StandardMaze2_Key	0.00 ± 0.00	0.06 ± 0.02	0.73 ± 0.11	**0.75 ± 0.08**	0.63 ± 0.14
StandardMaze3_Key	0.00 ± 0.00	0.30 ± 0.09	0.78 ± 0.07	0.84 ± 0.09	**0.94 ± 0.05**
Mean	0.03 ± 0.01	0.45 ± 0.01	0.84 ± 0.01	0.90 ± 0.03	**0.93 ± 0.02**

Key Minigrid 17x17 Results

Table 8: Key Minigrid 17x17 Solve Rate

Level	PLR - MNA	ACCEL - MNA	DEGen - MNA
SixteenRooms_Key	0.99 ± 0.01	0.86 ± 0.05	**1.00 ± 0.00**
SixteenRooms2_Key	0.86 ± 0.03	0.40 ± 0.13	**0.95 ± 0.05**
FourRooms_Key	0.80 ± 0.11	0.64 ± 0.09	**0.94 ± 0.05**
FourRooms2_Key	0.92 ± 0.05	0.81 ± 0.10	**0.95 ± 0.04**
FourRooms3_Key	0.88 ± 0.07	0.65 ± 0.09	**0.94 ± 0.04**
StandardMaze_Key	**0.69 ± 0.13**	0.19 ± 0.08	0.23 ± 0.05
StandardMaze2_Key	0.29 ± 0.09	0.21 ± 0.07	**0.50 ± 0.07**
StandardMaze3_Key	0.73 ± 0.09	**0.83 ± 0.06**	0.73 ± 0.09
Mean	0.77 ± 0.04	0.57 ± 0.03	**0.78 ± 0.03**

Key Minigrid 21x21 Results

Table 9: Key Minigrid 21x21 Solve Rate

Level	PLR - MNA	ACCEL - MNA	DEGen - MNA
SixteenRooms_Key	0.60 ± 0.11	0.60 ± 0.12	**1.00 ± 0.00**
SixteenRooms2_Key	0.40 ± 0.09	0.45 ± 0.11	**0.99 ± 0.01**
FourRooms_Key	0.21 ± 0.10	0.41 ± 0.11	**0.99 ± 0.01**
FourRooms2_Key	**0.93 ± 0.04**	0.59 ± 0.09	**0.93 ± 0.05**
FourRooms3_Key	0.35 ± 0.12	0.19 ± 0.08	**0.98 ± 0.02**
StandardMaze_Key	0.34 ± 0.11	0.06 ± 0.03	**0.56 ± 0.11**
StandardMaze2_Key	0.26 ± 0.12	0.00 ± 0.00	**0.39 ± 0.12**
StandardMaze3_Key	0.38 ± 0.11	0.16 ± 0.07	**0.61 ± 0.11**
Mean	0.43 ± 0.05	0.31 ± 0.04	**0.80 ± 0.03**

B Sokoban Environment

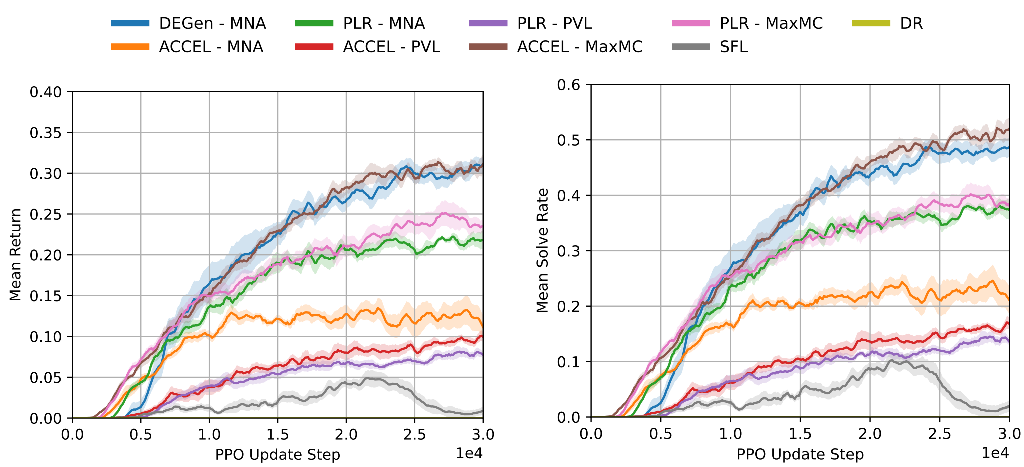
We have additionally performed experiments in a Sokoban-style environment. As in standard Sokoban, the agent aims to get all boxes to their storage locations. The agent receives a sparse reward when completing a level inversly proportional to how many steps were required to complete the level. For this domain, similarly to minigrid the agent has 5x5 observation space ahead of the agent, and an action space of move *forward*, turn *left* or turn *right*. The agent also has a *reset* action that resets the agent and boxes to their starting locations.

We used 9x9 levels for training - for DR, PLR and SFL, the random level generator generated levels that had 15 walls, and between 1 and 10 boxes. For DEGen, the generator could fill each newly observed cell in the environment with an empty square/wall/box/storage location/box on storage location. For all methods, we used identical hyperparameters to the minigrid environment (See Table 1).

B.1 Sokoban Results

From the results shown in Figure 8 and Tables 10 and 11, we see a large range in the performance of the various UED methods. We see that ACCEL using MaxMC is the best performing method, although comparable performance is achieved using DEGen. In Sokoban, we see that MaxMC outperforms MNA in the replay-based methods, with a substantial performance difference when used with ACCEL. Sokoban presents unique challenges compared to the previous environments used in this work. Primarily, the majority of randomly-sampled levels tend to be impossible, given that it only takes one box or storage location to be unreachable to make the entire level unsolvable. However, it is also likely that a high proportion of the solvable randomly-sampled levels will be difficult.

As all impossible levels will necessarily score zero with both MaxMC and MNA, We hypothesise that MaxMC may be a better metric than MNA in domains where difficult levels represent a higher proportion of the non-zero scoring randomly-sampled levels. The high performance of ACCEL in Sokoban is likely due to the clear difficulty scaling that can be achieved with ACCEL-like level evolution. In our implementation of ACCEL, one of the possible mutations is to add or remove a box/storage-location pair. This enables gradual difficulty evolution in Sokoban, which is less likely in the minigrid enviroments. As such, ACCEL is highly effective at generating an curriculum.



(a) Mean Return

(b) Mean Solve Rate

Figure 8: Sokoban zero-shot performance on hand-designed test set, showing mean and standard error across 8 runs.

We do however see in Tables 10 and 11 that a number of levels, those marked in *italics*, are not solved by any method. This suggests that there is room for future work to enable zero-shot performance on more difficult levels, and that Sokoban may be an interesting environment for future UED research.

B.2 Sokoban Zero-shot Transfer Levels

For the zero-shot transfer set, we have used the first 20 Sokoban Jr levels that do not exceed 13x13 in size.

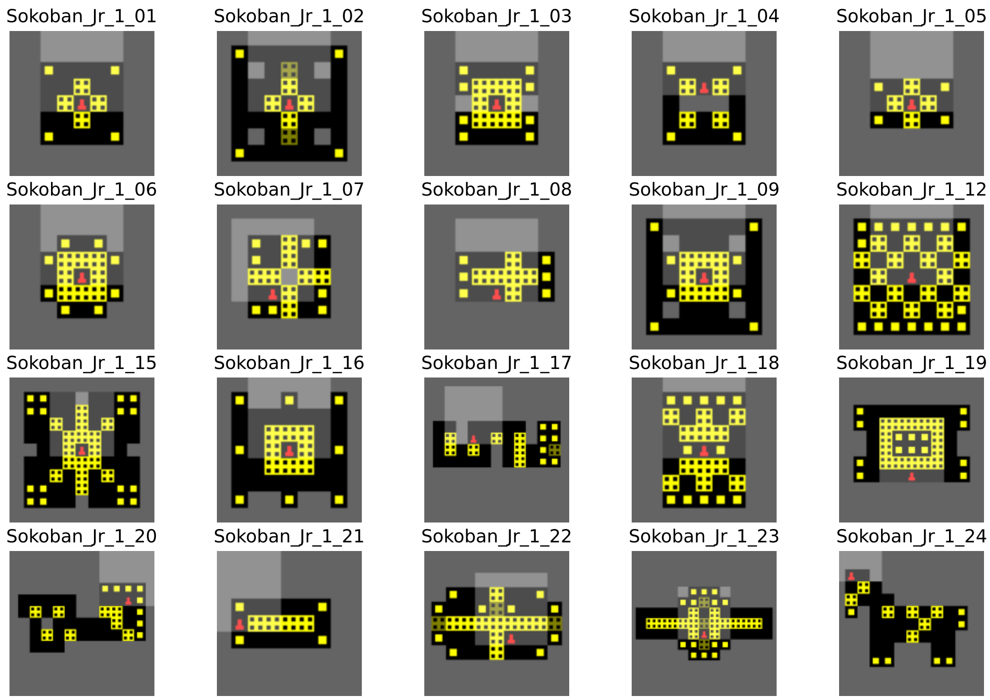


Figure 9: Hand designed evaluation levels for sokoban

B.3 Sokoban Tabular Results

Table 10: Sokoban Solve Rate (1)

Level	DR	SFL	PLR - MaxMC	ACCEL - MaxMC
Sokoban_Jr_1_01	0.00 ± 0.00	0.23 ± 0.15	**1.00 ± 0.00**	**1.00 ± 0.00**
Sokoban_Jr_1_02	0.00 ± 0.00	0.18 ± 0.12	0.94 ± 0.05	0.98 ± 0.02
Sokoban_Jr_1_03	0.00 ± 0.00	0.00 ± 0.00	0.46 ± 0.13	**0.83 ± 0.12**
Sokoban_Jr_1_04	0.00 ± 0.00	0.00 ± 0.00	0.16 ± 0.07	0.30 ± 0.09
Sokoban_Jr_1_05	0.00 ± 0.00	0.00 ± 0.00	0.42 ± 0.14	0.45 ± 0.12
Sokoban_Jr_1_06	0.00 ± 0.00	0.00 ± 0.00	0.74 ± 0.11	0.75 ± 0.11
Sokoban_Jr_1_07	0.00 ± 0.00	0.00 ± 0.00	0.60 ± 0.14	0.96 ± 0.03
Sokoban_Jr_1_08	0.00 ± 0.00	0.00 ± 0.00	0.21 ± 0.09	**0.50 ± 0.13**
Sokoban_Jr_1_09	0.00 ± 0.00	0.00 ± 0.00	0.65 ± 0.05	0.81 ± 0.04
Sokoban_Jr_1_12	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.29 ± 0.06
Sokoban_Jr_1_15	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_16	0.00 ± 0.00	0.00 ± 0.00	0.93 ± 0.04	**0.96 ± 0.02**
Sokoban_Jr_1_17	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_18	0.00 ± 0.00	0.00 ± 0.00	0.04 ± 0.03	0.46 ± 0.10
Sokoban_Jr_1_19	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.23 ± 0.11**
Sokoban_Jr_1_20	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_21	0.00 ± 0.00	0.00 ± 0.00	0.96 ± 0.03	**1.00 ± 0.00**
Sokoban_Jr_1_22	0.00 ± 0.00	0.00 ± 0.00	0.14 ± 0.02	**0.64 ± 0.10**
Sokoban_Jr_1_23	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_24	0.00 ± 0.00	0.00 ± 0.00	0.04 ± 0.03	**0.14 ± 0.05**
Mean	0.00 ± 0.00	0.02 ± 0.01	0.36 ± 0.01	**0.51 ± 0.02**

Table 11: Sokoban Solve Rate (2)

Level	PLR - PVL	ACCEL - PVL	PLR - MNA	ACCEL - MNA	DEGen - MNA
Sokoban_Jr_1_01	**1.00 ± 0.00**	0.99 ± 0.01	0.98 ± 0.02	0.89 ± 0.11	**1.00 ± 0.00**
Sokoban_Jr_1_02	0.24 ± 0.05	0.46 ± 0.06	0.98 ± 0.02	0.49 ± 0.07	**1.00 ± 0.00**
Sokoban_Jr_1_03	0.09 ± 0.07	0.25 ± 0.10	0.71 ± 0.11	0.15 ± 0.08	0.49 ± 0.14
Sokoban_Jr_1_04	0.00 ± 0.00	0.00 ± 0.00	**0.44 ± 0.09**	0.14 ± 0.08	0.35 ± 0.09
Sokoban_Jr_1_05	0.10 ± 0.10	0.01 ± 0.01	**0.58 ± 0.12**	0.48 ± 0.12	0.24 ± 0.16
Sokoban_Jr_1_06	0.05 ± 0.03	0.03 ± 0.02	0.50 ± 0.14	0.46 ± 0.12	**0.94 ± 0.03**
Sokoban_Jr_1_07	0.19 ± 0.13	0.51 ± 0.16	0.35 ± 0.09	0.19 ± 0.07	**1.00 ± 0.00**
Sokoban_Jr_1_08	0.00 ± 0.00	0.00 ± 0.00	0.15 ± 0.11	0.00 ± 0.00	**0.50 ± 0.15**
Sokoban_Jr_1_09	0.14 ± 0.04	0.13 ± 0.07	0.68 ± 0.06	0.18 ± 0.06	**0.85 ± 0.06**
Sokoban_Jr_1_12	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	**0.30 ± 0.15**
Sokoban_Jr_1_15	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_16	0.25 ± 0.05	0.13 ± 0.06	0.95 ± 0.03	0.60 ± 0.08	0.95 ± 0.03
Sokoban_Jr_1_17	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_18	0.00 ± 0.00	0.03 ± 0.02	0.00 ± 0.00	0.00 ± 0.00	**0.69 ± 0.14**
Sokoban_Jr_1_19	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.10 ± 0.06
Sokoban_Jr_1_20	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_21	0.51 ± 0.09	0.56 ± 0.12	0.76 ± 0.09	0.68 ± 0.12	0.99 ± 0.01
Sokoban_Jr_1_22	0.00 ± 0.00	0.06 ± 0.05	0.09 ± 0.05	0.00 ± 0.00	0.30 ± 0.11
Sokoban_Jr_1_23	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*	*0.00 ± 0.00*
Sokoban_Jr_1_24	0.00 ± 0.00	0.01 ± 0.01	0.05 ± 0.04	0.00 ± 0.00	0.09 ± 0.03
Mean	0.13 ± 0.01	0.16 ± 0.01	0.36 ± 0.01	0.21 ± 0.02	0.49 ± 0.02

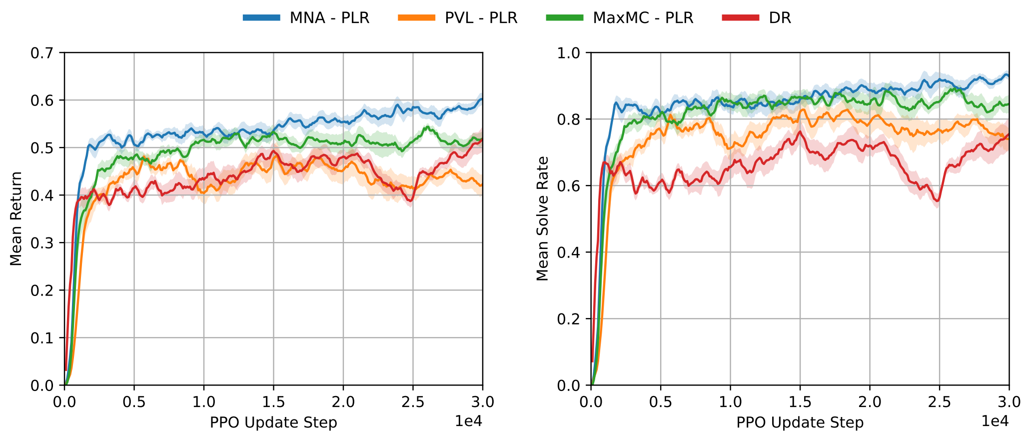
C Additional Results

C.1 MNA and Existing Methods

To directly examine the effectiveness of MNA compared to existing regret metics, we show zero-shot performance of ACCEL and PLR using MNA, PVL and MaxMC.

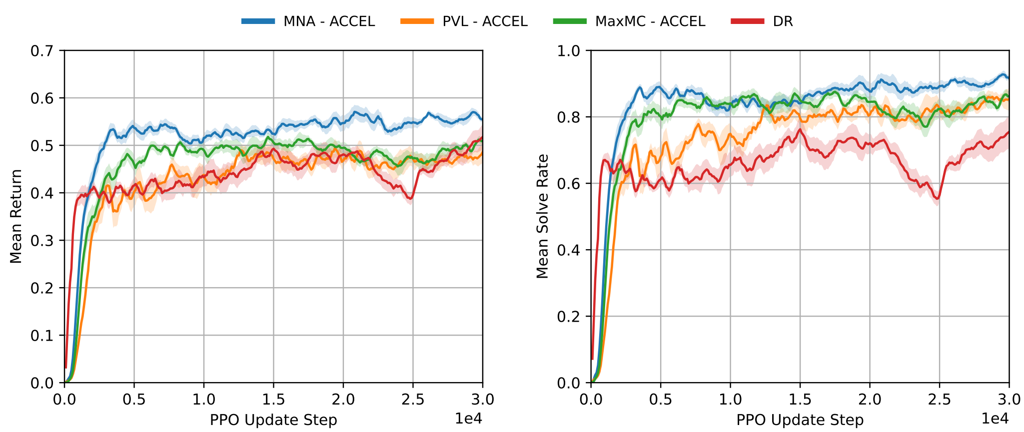
Minigrid

In Figures 10 and 11, we illustrate the relative performance of each of these metrics in the standard minigrid domain and show that MNA clearly outperforms other metrics, whether using either PLR or ACCEL.



(a) Mean Return (b) Mean Solve Rate

Figure 10: Minigrid - comparison of PLR performance using different metrics

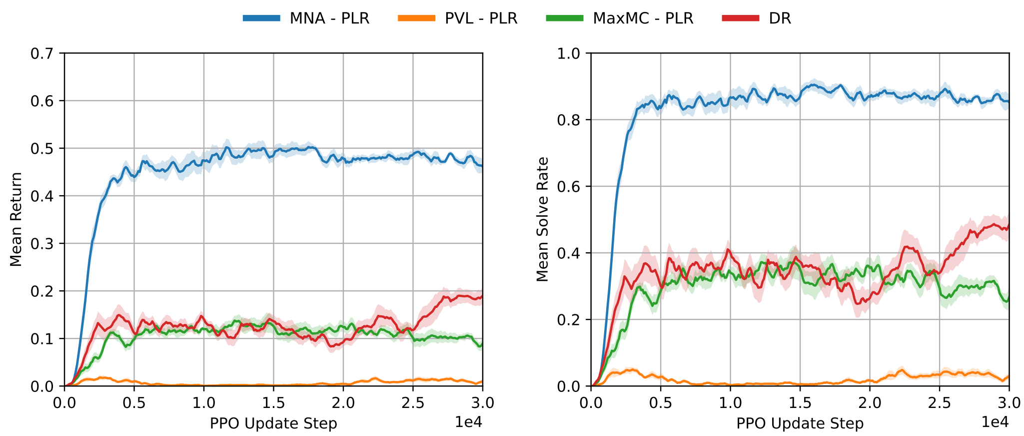


(a) Mean Return (b) Mean Solve Rate

Figure 11: Minigrid - comparison of ACCEL performance using different metrics

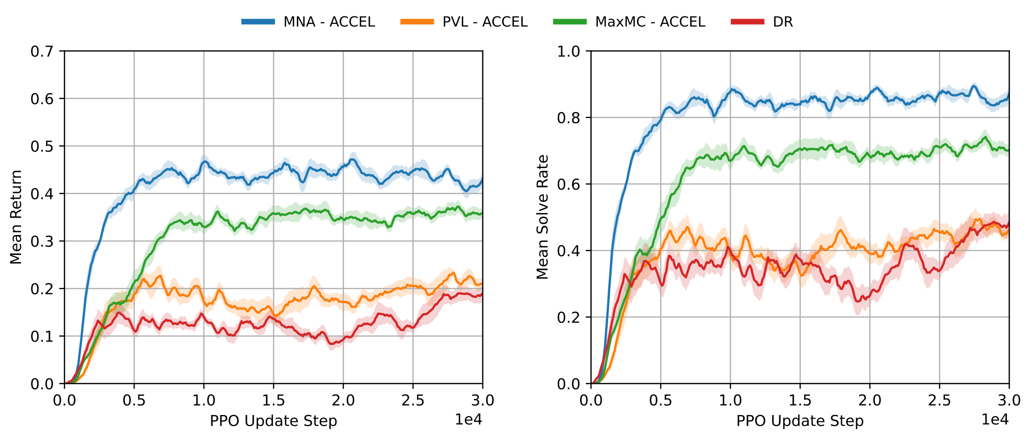
Key Minigrid

In Figures 12 and 13, we compare the same methods but on the key minigrid domain instead. Here, we again see that MNA outperforms existing methods - including a substantial performance improvement when using PLR.



(a) Mean Return (b) Mean Solve Rate

Figure 12: Key Minigrid - comparison of PLR performance using different metrics

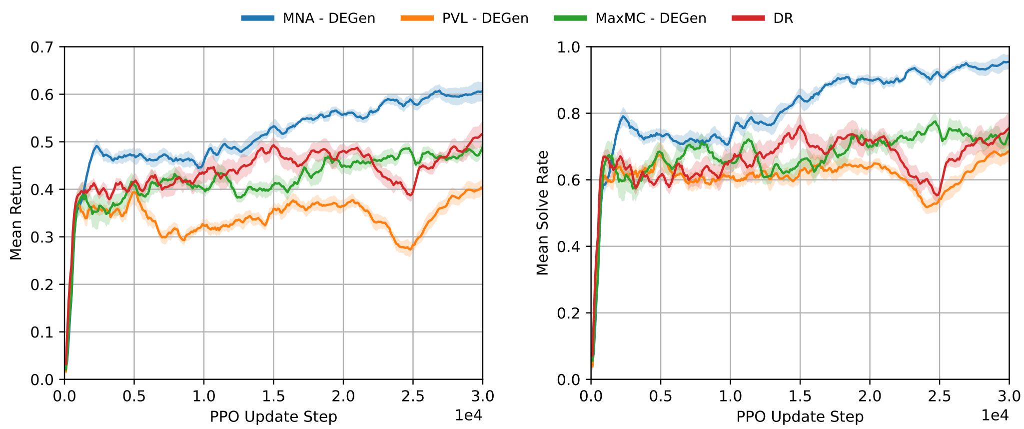


(a) Mean Return (b) Mean Solve Rate

Figure 13: Key Minigrid - comparison of ACCEL performance using different metrics

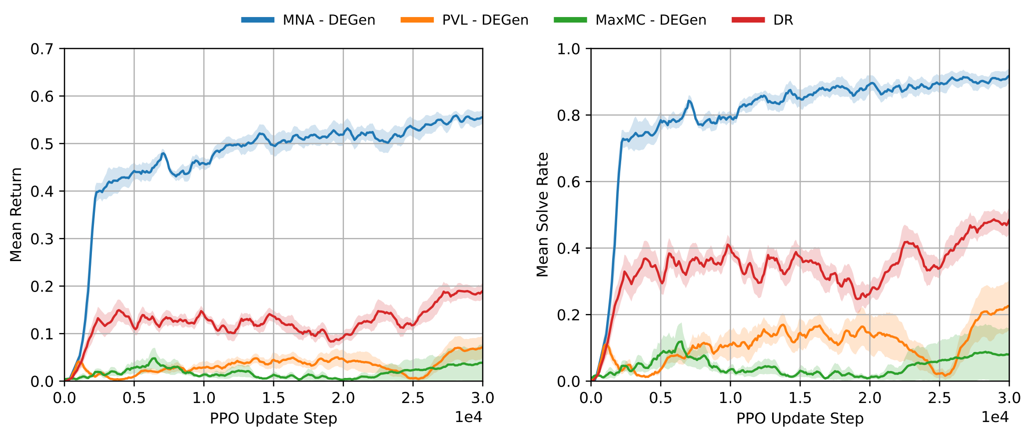
C.2 DEGen and Existing Regret Approximations

In order to illustrate the ineffectiveness of existing regret approximations when used as optimisation objectives for training a teacher, we show the relative performance of DEGen using MNA, PVL and MaxMC. Figures 14 and 15 show that MNA consistently outperforms PVL and MaxMC. We also see here that using a teacher trained using PVL and MaxMC results in at best equivalent, but generally worse, performance compared to naive domain randomisation.



(a) Mean Return (b) Mean Solve Rate

Figure 14: Minigrid - comparison of DEGen performance using different metrics

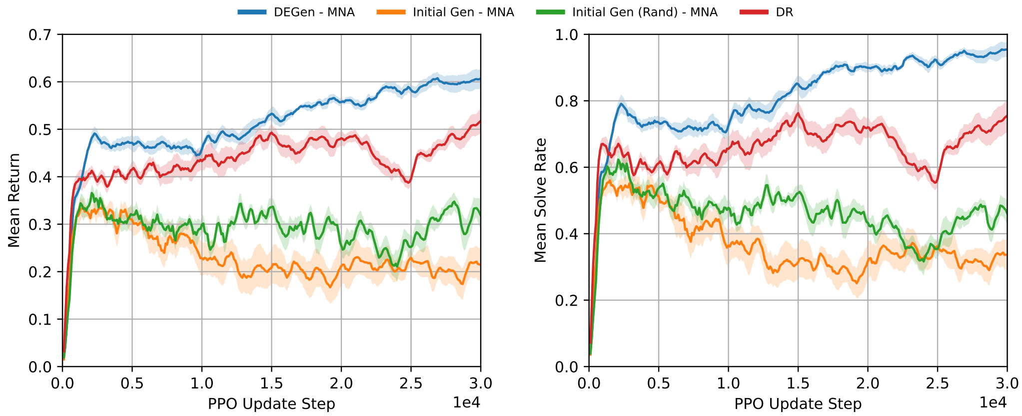


(a) Mean Return (b) Mean Solve Rate

Figure 15: Key Minigrid - comparison of DEGen performance using different metrics

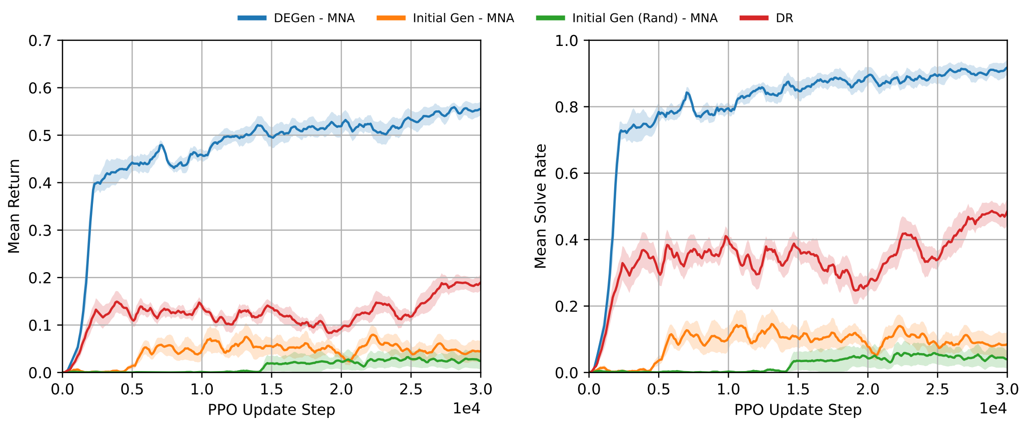
C.3 DEGen vs Initial Gen

In Figures 16 and 17, we show the performance of DEGen compared to the performance of a generator that generates the full level prior to student rollouts. We include both a standard level generator *Initial Gen*, identical to the PAIRED generator in JaxUED [8], as well as *Initial Gen (Rand)*, which randomly places the agent in the level before the rest of the level is constructed. We see that both methods performance worse than domain randomisation.



(a) Mean Return (b) Mean Solve Rate

Figure 16: Minigrid - comparison of Initial Gen and DEGen

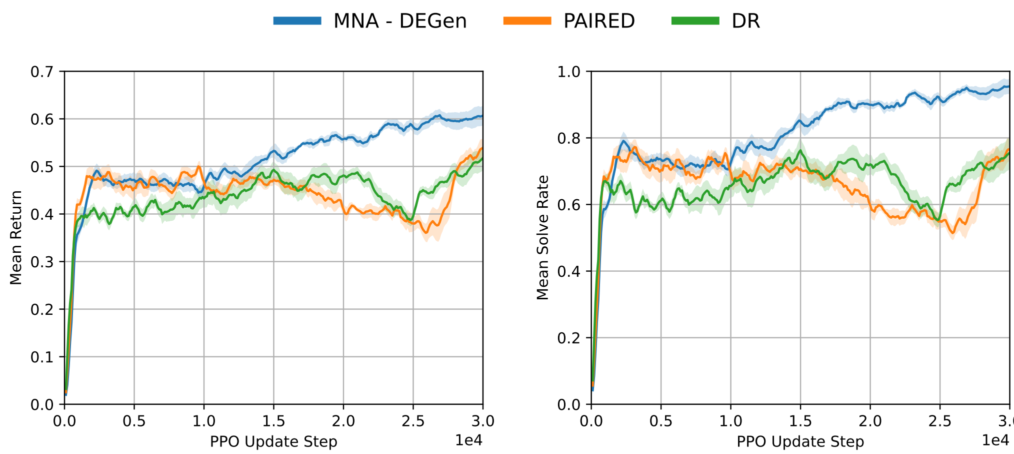


(a) Mean Return (b) Mean Solve Rate

Figure 17: Key Minigrid - comparison of Initial Gen and DEGen

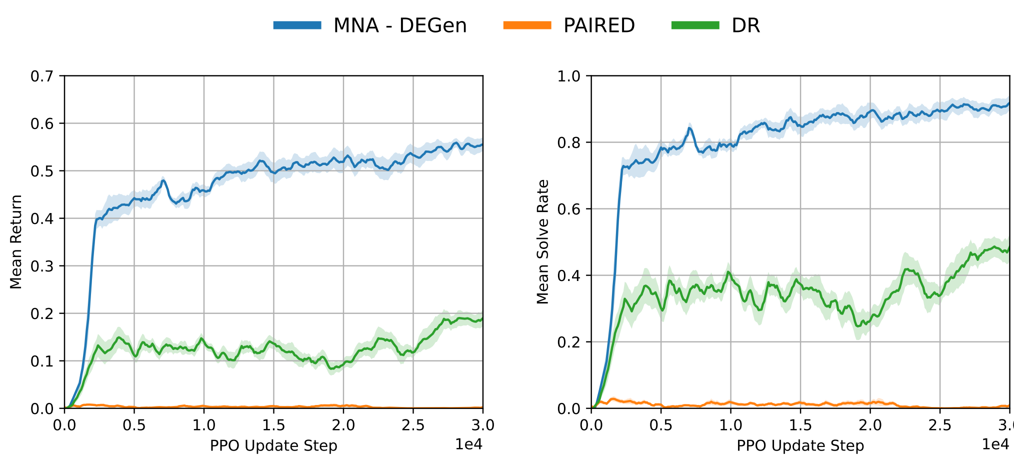
C.4 PAIRED

Finally, we examine the performance of DEGen compared to PAIRED [9]. In standard minigrid, we see that PAIRED performs very similarly to domain randomisation, and worse than DEGen. Additionally, we see in the key minigrid domain the limitations of the PAIRED regret approximation. As PAIRED relies on the antagonist's performance to approximate the best possible level return, high scoring levels require the antagonist to perform well. However, if a level is challenging due to some obstacle the student has not previously encountered, it is likely that the antagonist will also perform poorly, given it has been trained on the same set of levels as the student. Therefore, the PAIRED generator is unlikely to generate levels requiring the antagonist to use the key, and so as the student agent has not encountered levels similar to the zero-shot hand-designed levels that require the key, zero-shot performance is extremely poor.



(a) Mean Return (b) Mean Solve Rate

Figure 18: Minigrid - comparison of PAIRED and DEGen



(a) Mean Return (b) Mean Solve Rate

Figure 19: Key Minigrid - comparison of PAIRED and DEGen

C.5 Training Level Examples

We have included examples of levels generated by each method in the repository at `https://github.com/HarryMJMead/Dynamic-Environment-Generation-for-UED`. These levels were all sampled from the final training step.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract are accurately reflected in the description of our method and our results.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have discussed the limitations of this work in Section 8.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: This paper contains no theoretical results.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.

- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The full method used is outlined in Sections 3 and 4, and further detail is provided in Appendix A

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We include the code and instructions in the supplementary material

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All addtional hyperparameters and experiment details are included in Appendix A.2

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All results are taken over multiple seeds with standard error

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Details on compute and time are provided in Appendix A

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Our work is not expected to have direct societal impacts.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our paper does not present any of these risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All prior work that has been used has been cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We have provided details on training, as well as code and documentation.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our work does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This work does not involve research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLMs were not used in any important, original or non-standard components of this reserach.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.